DEFENCE R&D DÉFENSE

The Convair 580 Along-Track INSAR Processor

Documentation, Guide and History

Ishuwa Sikaneta and John Campbell
Defence Research Establishment Ottawa

Mike Robson
MDA

20020110 043

Defence R&D Canada

DEFENCE RESEARCH ESTABLISHMENT OTTAWA

TECHNICAL MEMORANDUM
DREO TM 2001-053
August 2001

National Defence Défense nationale

Canada

DEFENCE DÉFENSE

The Convair 580 Along-Track INSAR Processor

Documentation, Guide and History

Ishuwa Sikaneta and John Campbell
Surveillance Radar Group
Aerospace Radar and Navigation Section

Mike Robson
MDA

DEFENCE RESEARCH ESTABLISHMENT OTTAWA

TECHNICAL MEMORANDUM
DREO TM 2001-053
August 2001

Project
05EG11

AQ F02-03-0454

Abstract

This document summarizes the work done on the along-track interferometric SAR processor loaned to Defence Research Establishment Ottawa (DREO) by the Canada Centre for Remote Sensing (CCRS). It outlines the problems encountered with getting the software to work properly, as well as the enhancements that were made to the code. It provides a guide to setting up and using the processor. It details the changes in structure made to the program , and describes some of the theoretical ideas behind changes and enhancements. It provides, finally, a document deliverable to CCRS that fulfills part of the loan agreement.

Résumé

Le présent document résume le travail effectué sur le processeur de RAS interférométrique longitudinal, prêté par le Centre canadien de télédétection (CCT) au Centre de recherches pour la défense Ottawa (CRDO). Il décrit brièvement les problèmes qu'on a rencontrés en essayant de bien faire fonctionner le logiciel et expose à grands traits les améliorations apportées au code. Il fournit un guide de mise au point et d'exploitation du processeur. Il décrit en détail les changements structuraux subis par le programme et explique certaines des idées théoriques qui sont à la base des changements et des améliorations. Enfin, il fournit un document à livrer au CCT, ce qui satisfait à une partie de l'accord de prêt.

This page intentionally left blank.

Executive summary

The Convair 580 along-track antenna system is being used as an airborne experimental system in support of the RADARSAT 2 Ground Moving Target Indicator (GMTI) investigation. The main aim of the along-track system is to detect temporal displacements, usually a result of the velocity/acceleration of a target.

The synthetic aperture raw data produced by the Convair is focussed to an image by using the program *atinsar* provided by CCRS. Once the data is processed for the two phase centers of the along-track antenna, an interferogram is produced, and the phase difference between the two channels is used as an indication of the presence of, and a measure of, the motion of a target.

The processing software was outdated. One of the first objectives was to make a port from Irix 5.x to the Irix 6.5 operating system. The software was also inefficient so the second objective was to implement the SGI specific routines.

Since the objective was to detect moving targets, the definition of the processing kernel was modified to allow for hypothetical moving targets. As a result, the detection of moving targets was enhanced, and the estimation of the velocity or acceleration of moving targets was also improved.

Modifications to the processor introduced further complexity to the software, and since there was not any documentation to begin with, a guide was compiled to help with both running and compiling the software.

Although the speed, accuracy and flexibility of the processor has been improved, there are still many changes that would make the processor a more useful experimental tool. First, by further modifying the structure of the program, repetitive tasks can be made to occur only once. Second, by making use of current computer technology such as dual processing, speed can be increased. Third, by introducing flexibility into detection and focussing filters, more varied types of datasets can be investigated. Fourth, by building a range walk compensation filter, accuracy can be improved and calibration of moving targets can be properly addressed. Finally, the option of automatic detection, measurement and classification of moving targets can be built into to the processor.

Ishuwa Sikaneta , John Campbell , Mike Robson . 2001. The Convair 580 Along-Track INSAR processor. TM 2001-053. Defence Research Establishment Ottawa.

Sommaire

Le système d'antenne longitudinale Convair 580 sert actuellement de système expérimental aéroporté, contribuant à la recherche effectuée par l'indicateur de cible terrestre mobile (GMTI), RADARSAT 2. Le système longitudinal sert principalement à détecter des déplacements temporels, résultant généralement de la vitesse ou de l'accélération d'une cible.

Les données brutes pour ouverture synthétique produites par le Convair sont focalisées pour former une image au moyen du programme atinsar fourni par le CCT. Une fois qu'on a traité les données relatives aux deux centres de phase de l'antenne longitudinale, on enregistre un interférogramme et on se base sur la différence entre les deux canaux pour déterminer si la cible est mobile et mesurer le mouvement.

Le logiciel de traitement était désuet. On avait donc décidé, comme un des premiers objectifs, à créer un port du système Irix 5.x au système d'exploitation Irix 6.5. Le logiciel était d'autre part inefficace, aussi le deuxième objectif consistait-il à appliquer les routines particulières de la SGI. Puisque le but était de détecter des cibles mobiles, on a modifié la définition du noyau de traitement pour prévoir des cibles mobiles hypothétiques. Cela a amélioré non seulement la détection mais aussi l'évaluation de la vitesse/de l'accélération des cibles mobiles.

Les modifications apportées au processeur a rendu le logiciel plus complexe et puisqu'il n'y avait aucune documentation pour commencer, on a compilé un guide qui facilite à la fois l'exploitation et la compilation du logiciel.

Bien que le processeur soit devenu plus rapide, plus précis et plus souple, il faudrait encore y apporter beaucoup de changements pour qu'il devienne un outil d'expérimentation. Premièrement, on peut modifier davantage la structure du programme de faon que les tches répétitives ne s'effectuent qu'une seule fois. Deuxièmement, on peut augmenter la vitesse en utilisant la technologie moderne du traitement de l'information. Troisièmement, on peut rendre les filtres de détection et les filtres focalisants plus flexibles, ce qui permettra d'étudier des données beaucoup plus variées. Quatrièmement, une série de filtres à compensation améliorerait l'exactitude et permettrait d'effectuer l'étalonnage des cibles plus convenablement. Enfin, on peut incorporer l'option de détection, de mesure et de classification automatiques dans le processeur.

Ishuwa Sikaneta , John Campbell , Mike Robson . 2001. . TM 2001-053. .

Table of contents

List of figures

List of tables

1. Introduction

The Canada Centre for Remote Sensing (CCRS) loaned an Along-Track Interferometric SAR (ATINSAR) processor to the Defence Research Establishment of Ottawa (DREO). Under the terms of the agreement, DREO was obliged to return to CCRS, all updated software and full documentation of changes made, including enhancements, to the processor. Along with the updated program, this report is the first effort at fulfilling that obligation.

The purpose of the loan was to provide DREO with the capability to conduct Ground Moving Target Indicator (GMTI) experiments with data collected by the Department of the Environment's Convair 580. Any changes or enhancements that have been made thus center around aspects of the processor designed to detect ground moving targets. In addition, changes have been made that improve the speed and accuracy of the processor, the readability of the code and the modularity of the program. Finally, the processor has been modified to compile specifically on SGI Octane systems running IRIX6.5 in 64-bit mode.

Although the processor was intended only as an experimental tool, all modifications have been designed to facilitate further upgrade.

In light of the fact that the program was authored by several individuals, and that documentation is fragmented, this report can also serve as a guide to anyone wishing to investigate the program.

During the time that the program has been at DREO, three individuals have worked on the code. The first was John Campbell working for DREO, the second was Mike Robson working for McDonald Detwiller and Associates (MDA) and finally Ishuwa Sikaneta with DREO.

This document starts by describing the delivered product and outlines changes that have been made to the old processor. It then describes how the new processor is compiled and how it is run. Next, the document goes into more detail on the changes that were made to the old processor. The sixth section can be considered as detailed documentation on the code for both the old and new processors. In the second last section, some of the theory behind changes made to the new processor is described, and some ideas for future enhancements are presented. Finally, the last section highlights the investment that DREO has made into the new processor and identifies DREO intellectual property.

2. Delivered product

The main engine of the processor is written in C, but MATLAB is used to preprocess some of the motion compensation data. The delivered product therefore includes the main C-program, various utility C-programs and numerous MATLAB routines in addition to the documentation.

2.1 Outline of Major Changes

2.1.1 Code Documentation

Efforts were made to insert, directly in the code, comments on all of the changes that were made. In addition, comments that describe the purpose of many segments of the code were added.

2.1.2 Full, Flexible Reference Function

The full reference function was built into the program. Not only was the complete hyperbolic function substituted for the parabolic approximation, but the flexibility of defining a model for target motion was also built.

2.1.3 Command Line Arguments

Command line arguments were built into *atinsar1.4*. The command line now reads:

atinsarnew [-options] <paramfile> [-options], [1]

where *options* are one of those listed in table 1.

2.1.4 MATLAB Version Compatibility

The MATLAB routines and file formats are fully compatible with MATLAB version 5.x

[1]In passing a command line argument that includes an argument such as *-SLC=test.img* care must be taken to make sure that there are no spaces in the expression *i.e.* *-SLC=_test.img* will not work.

option	function
-h	Show a help screen
-help	Show a help screen
-v	Operate in verbose mode
-workdir=	define a directory that contains the data if not the current directory
-f	Output the co-registered fore and aft channels separately
-s	Output a SLC flat file
-slc=	Output a named flat file
-r1dot=	Use a radial velocity offset for processing
-a1dot=	Use an azimuth velocity offset for processing
-z1dot=	Use a vertical velocity offset for processing
-r2dot=	Use a radial acceleration offset for processing
-a2dot=	Use an azimuth acceleration offset for processing
-z2dot=	Use a vertical acceleration offset for processing

Table 1: *Command line arguments for atinsar1.4*

2.1.5 Data Capability

The data volume capability of the processor has been increased. The program has been compiled in 64-bit mode, so the 2 gigabyte data file size limit is no longer a factor.

3. Compile HOW-TO

The present version of the software is designed to dynamically link to required libraries at run-time. Therefore, in addition to compiling the program itself, it is necessary to compile all libraries upon which the program depends. Furthermore, MATLAB preprocessing depends upon .mex files which may also have to be compiled.

3.1 Required files

In this HOW-TO, we are concerned only with compiling the program *atinsar1.4*. Other C-programs which are part of ATINSAR are assumed to have already been compiled and to be available. Table 2 lists all of the other C-programs which are part of the software package. Other than the

sigdmp2
ancmat2
meandati
insarmaid
gpsmat
qcnati

Table 2: *Other C-programs in the software package*

aforementioned C-programs, ATINSAR is also composed of MATLAB **.m** files as listed in table 3 and MATLAB **.mex** files as listed in table 4.

root_dir/MATLAB/startATI.m
root_dir/MATLAB/gpspro.m
root_dir/MATLAB/atimocnew.m
root_dir/MATLAB/atitimetestnew.m
root_dir/MATLAB/atiprf.m
root_dir/MATLAB/landphscal.m
root_dir/MATLAB/timecode.m

Table 3: *Required MATLAB .m files and suggested location*

The MATLAB .m files do not need to be compiled.

The MATLAB .mex files do need to be compiled. They each have a corresponding source file, and some of them depend further upon a header file.

root_dir/mex/append.mexsg64
root_dir/mex/argmscan.mexsg64
root_dir/mex/argmsubs.mexsg64
root_dir/mex/argmwrte.mexsg64
root_dir/mex/cspline.mexsg64
root_dir/mex/new_filename.mexsg64
root_dir/mex/newm_filename.mexsg64
root_dir/mex/sigtime.mexsg64
root_dir/mex/smooft.mexsg64
root_dir/mex/trackcal.mexsg64
root_dir/mex/wgs84px.mexsg64

Table 4: *Required MATLAB .mex files and suggested location*

Table 5 lists all of the files as well as a suggested location for each file. Finally, the source files for all of the dynamic link libraries and program are needed. They are listed in table 6 along with a suggested location and the name of the library/program that they build.

3.2 MATLAB libraries

The MATLAB libraries libmat.so and libmx.so are required at run-time. At compile time, the MATLAB distributed file mat.h is required.

3.3 MEX files

The .mex files are compiled from within MATLAB. A C-compiler will **have** to be on the system in order for the MATLAB compiler to work. [2]

The .mex file compilation is done, for example, with the following command at the MATLAB prompt:

.mex -I/root_dir/include append.c

The *-I* flag tells the compiler where to look for include files. All of the files listed as .mex files will have to be compiled. The output from the above command is a file in the current directory named *append.mexsg64.*

[2]The MATLAB compiler piggy backs off the system compiler.

root_dir/mex/append.c
root_dir/mex/argmscan.c
root_dir/mex/argmsubs.c
root_dir/mex/argmwrte.c
root_dir/mex/argsubs.c
root_dir/mex/cspline.c
root_dir/mex/new_filename.c
root_dir/mex/newm_filename.c
root_dir/mex/sigtime.c
root_dir/mex/smooft.c
root_dir/mex/trackcal.c
root_dir/mex/wgs84px.c
root_dir/include/mapproj.h
root_dir/include/integers.h
root_dir/include/utility.h
root_dir/include/argsubs.h
root_dir/include/argmsubs.h

***Table 5:** Required MATLAB .mex source files and suggested location*

3.4 Makefiles

Makefiles for each of the libraries and for the program itself will have to be tailored to suit the structure of the system on which the build is taking place. If the suggested directory structure is used, then the only change that will have to be made is in the value of the ROOTPATH variable (e.g. ROOTPATH=root_dir).

For each of the library Makefiles, *make install64* will compile and install the library in the *lib64* directory. For the actual program itself, *make install* will compile and install the program in the *bin* directory. The root *lib64*, *lib32*, and *bin* directories should exist.

Although *atinsar1.4* is compiled as a 64-bit program, there exists also the option of installing the 32-bit versions of the libraries, *make install32*.

For each of the Makefiles, *make clean* will remove all object, library or program files in the make directory.

3.5 Environment variables

Some environment variables will have to be set at run-time. These variables, LD_LIBRARY64_PATH, PATH and MATLABPATH are discussed further in

root_dir/src/matlabio/Makefile	libc_matio64.so
root_dir/src/matlabio/getmatlab_vector.c	libc_matio64.so
root_dir/include/getmatlab_vector.h	libc_matio64.so
root_dir/src/c_shared/Makefile	libc_shared.so
root_dir/src/c_shared/utility.c	libc_shared.so
root_dir/src/c_shared/local_error.c	libc_shared.so
root_dir/src/c_shared/argsubs.c	libc_shared.so
root_dir/src/c_shared/readsuni.c	libc_shared.so
root_dir/src/c_shared/chcksub2.c	libc_shared.so
root_dir/include/integers.h	various
root_dir/include/local_error.h	libc_shared.so
root_dir/include/readsuni.h	libc_shared.so
root_dir/include/chcksub2.h	libc_shared.so
root_dir/include/utility.h	various
root_dir/include/argsubs.h	various
root_dir/src/numerical/Makefile	libc_numerical.so
root_dir/src/numerical/spline2.c	libc_numerical.so
root_dir/src/numerical/convlv_sgi.c	libc_numerical.so
root_dir/src/numerical/vectors.c	libc_numerical.so
root_dir/src/numerical/downgaus.c	libc_numerical.so
root_dir/include/vectors.h	libc_numerical.so
root_dir/include/downgaus.h	libc_numerical.so
root_dir/include/spline2.h	libc_numerical.so
root_dir/include/convlv_sgi.h	libc_numerical.so
root_dir/src/atinsar1.4/atinsar1.4.c	atinsar1.4

Table 6: *Required C-source files, suggested location and the library/program that they build*

the HOW-TO on using the processor.

4. Processor HOW-TO

4.1 Introduction

This note is intended as a guide on how to run the *atinsar1.4* airborne SAR processor. It is designed to take a new user from dumping signal data to producing a final calibrated interferogram. Along the way, useful information regarding the purpose of each step will be provided.

4.2 Program Setup

The *atinsar1.4* processor has been compiled and designed to run on an IRIX 6.5 system. There are no ports to other platforms at present.

The software will need to be made known to the PATH and LD_LIBRARY64_PATH environment variables. In addition, MATLAB paths will need to be set up. All files and programs listed in table 7 will need to be on the system.

Type of file	Suggested Location/File Name
Executable	root_dir/bin/atinsar1.4
Executable	root_dir/bin/qcnati
Executable	root_dir/bin/ancmat2
Executable	root_dir/bin/sigdmp2
Executable	root_dir/bin/meandati
Executable	root_dir/bin/insarmaid
Executable	root_dir/bin/gpsmat
Library	root_dir/lib64/libc_numerical
Library	root_dir/lib64/libmatio.co
Library	root_dir/lib64/libc_shared.so
M File	root_dir/MATLAB/startATI.m
M File	root_dir/MATLAB/gpspro.m
M File	root_dir/MATLAB/atimocnew.m
M File	root_dir/MATLAB/atitimetestnew.m
M File	root_dir/MATLAB/atiprf.m
M File	root_dir/MATLAB/landphscal.m
M File	root_dir/MATLAB/timecode.m
.mex	root_dir/mex/append.mexsg64
.mex	root_dir/mex/argmscan.mexsg64
.mex	root_dir/mex/argmsubs.mexsg64
.mex	root_dir/mex/argmwrte.mexsg64
.mex	root_dir/mex/cspline.mexsg64
.mex	root_dir/mex/new_filename.mexsg64
.mex	root_dir/mex/newm_filename.mexsg64
.mex	root_dir/mex/sigtime.mexsg64
.mex	root_dir/mex/smooft.mexsg64
.mex	root_dir/mex/trackcal.mexsg64
.mex	root_dir/mex/wgs84px.mexsg64

Table 7: *Files that need to be installed on the system and suggested location*

A suggested location for each file has been given. For the executable files, it will be necessary to update the PATH variable as well as the LD_LIBRARY64_PATH variable. The PATH variable can be updated by either typing (on the command line), or adding to your .cshrc file, the following:

```
setenv PATH root_dir/bin:$(PATH)
```

The LD_LIBRARY64_PATH environment variable can be updated by either typing the following on the command line

```
setenv LD_LIBRARY64_PATH root_dir/bin:$(LD_LIBRARY64_PATH),
```

or by adding the following lines to your .cshrc file

```
if(${?LD_LIBRARY64_PATH}) then
        setenv LD_LIBRARY64_PATH root_dir/lib:$(LD_LIBRARY64_PATH)
else
        setenv LD_LIBRARY64_PATH root_dir/lib
endif
```

The MATLAB path specification may be done either in your .cshrc file (by updating the MATLABPATH environment variable as above), or in your startup.m file. By adding the following lines

```
addpath root_dir/MATLAB/
addpath root_dir/mex/
```

to your startup.m file.

Once all of the files listed in table 7 have been installed and the appropriate environment variables have been updated, processing of data files may begin.

4.3 Process chain

Figure 4.3 shows a schematic of the processing procedure.

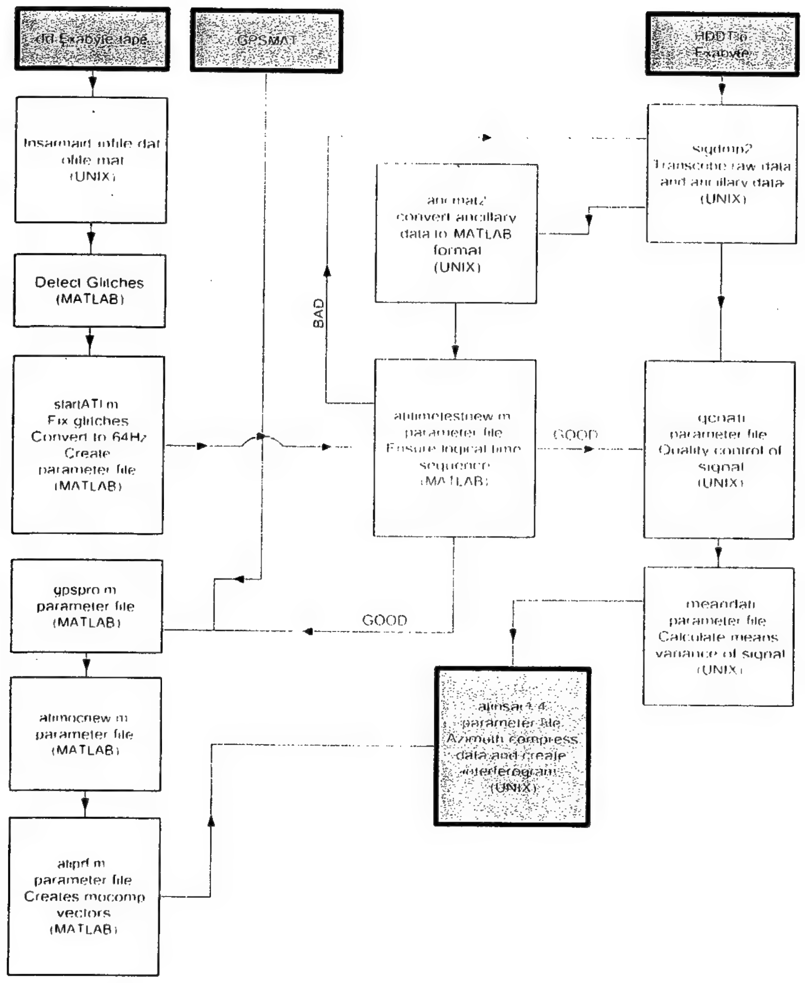

dd Exabyte tape
GPSMAT
HDDT to Exabyte
Insarmaid infile dat ofile mat (UNIX)
sigdmp2 Transcribe raw data and ancillary data (UNIX)
ancmat2 convert ancillary data to MATLAB format (UNIX)
Detect Glitches (MATLAB)
BAD
startATI m Fix glitches Convert to 64Hz Create parameter file (MATLAB)
attimetestnew m parameter file Ensure logical time sequence (MATLAB)
GOOD
qcnati parameter file Quality control of signal (UNIX)
gpspro m parameter file (MATLAB)
GOOD
meandati parameter file Calculate means variance of signal (UNIX)
atimocnew m parameter file (MATLAB)
atinsar4.4 parameter file Azimuth compress data and create interferogram (UNIX)
atipd m parameter file Creates mocomp vectors (MATLAB)

Figure 1: Airborne SAR processor chain

4.4 Processing Overview

There are really four steps involved in data processing:

1. Load signal and mocomp data onto the computer,
2. Preprocess mocomp data and signal data, then set up a program parameter file,
3. Run the program with program parameter file,
4. Calibrate the interferogram.

4.5 One: How to load the data

The first step is to load data onto the computer. There are a number of data files required by the processor. Table 8 lists all of the data files that are required for a hypothetical line 8, pass 9 dataset. For the remainder of this guide, it will be assumed that line 8 pass 9 is being processed.[3]

File	Source
l8p9ca.sig	from raw data exabyte tape
l8p9cb.sig	from raw data exabyte tape
l8p9xa.sig	from raw data exabyte tape
l8p9xb.sig	from raw data exabyte tape
l8p9ca.anc	from raw data exabyte tape
l8p9cb.anc	from raw data exabyte tape
l8p9xa.anc	from raw data exabyte tape
l8p9xb.anc	from raw data exabyte tape
l8p9ca.hdr	from raw data exabyte tape
l8p9cb.hdr	from raw data exabyte tape
l8p9xa.hdr	from raw data exabyte tape
l8p9xb.hdr	from raw data exabyte tape
l8p9maid.dat	from mocomp maid file exabyte tape
l8p9gps.dat	from gps processor of gps data

***Table 8:** Data files that are required for ATINSAR processing*

[3]The symbol lxpx really means line x pass x where the x's are any numbers, like 8 and 9.

4.5.1 Signal data

The raw data exabyte tape is produced by stripping data from a larger tape that usually contains a much larger set of radar signal data from a given flight. The larger tape, a SONY Helical Scan tape, is produced on board the aircraft. The process of extracting data from the helical scan tape will not be discussed in this guide.

Signal data, along with other critical information, has to be extracted from the exabyte tape. The program *sigdmp2* is designed to extract the data and sort it into appropriate files. The name of your local tape drive will need to be known in advance. The program *sigdmp2* is pretty straight-forward to run because it is interactive. A couple of things to keep in mind are that the tape identification signature should ideally reflect the line and pass that are being processed so one might enter l8p9ca as the tape identification for the first file on tape. The files will probably be stored consecutively on tape in the order ca, cb, xa, xb.[4] Another thing to keep in mind is that when asked by the program, both the signal and ancillary data should be dumped.

Typically, all lines and samples from a pass are dumped, but a subsection can be dumped if the subsection parameters are known in advance.

The program *sigdmp2* will create the (.sig, .anc and .hdr) files listed in table 8. Write permission, of course, will be required in the directory where the program is run.

Prior to November 1994, only a subset of ancillary parameters that are now available (Oct, 2000) could be dumped from tape. The first version of *sigdmp* was created to read the limited set of data. Now, however, with a larger ancillary data set, *sigdmp2* is required.

4.5.2 Maid Data

The mocomp maid exabyte tape is shipped from the aircraft directly. The tape will contain maid data from all passes on the flight. The data will have to be extracted from tape and put onto the computer hard disk. There does not seem to be any standard method for extracting the data from tape. The following procedure, however, is one way of doing so.

[4]The order of the files on tape is not standard and the individual who wrote the data onto exabyte tape should be consulted.

1. Use dd to extract a file from the tape
   dd if=/dev/rmt/tps1d6nrnsv.8500 of=file01.dat bs=1024b[5]

2. Repeat step 1 until the output from dd **consistently** (3 or four times) shows that 0 bytes of data are being read from the tape. Make sure to update the output file name in step one, i.e. to file02.dat.

It will be necessary to find out which extracted file corresponds to which pass. The data will have to be converted into MATLAB format before this determination can be done. The program that converts maid data into MATLAB format is called *insarmaid.* The following command will do the conversion:

insarmaid filexx.dat filexx.mat

To then determine which file belongs to which pass, open the MATLAB file in MATLAB (start MATLAB, type load filexx.mat), and examine the time vector by typing timecode(time(1)) at the MATLAB prompt. Comparison between the time returned by timecode with the time that each pass was acquired will show which file belongs to which pass. Once the file has been correlated to a pass, the file name should be changed so that it reflects the pass (i.e. for maid data from line 8 pass 9, the standard is to type: *mv filexx.mat l8p9maid.mat*)

4.5.3 GPS Data

This guide will not discuss GPS preprocessing. It will be assumed that the GPS data is on the system, either processed by Ashtec office or some other GPS processing program. As with the maid data, the GPS data will have to be converted into MATLAB format. The program *gpsmat* is designed to do the conversion:

gpsmat rawfile.dat l8p9gps.mat

4.6 Two: How to preprocess the mocomp and signal data

Preprocessing is, perhaps, the most user interactive step in the data processing chain. Both the mocomp and the signal data need to be preprocessed. Although the sequence of steps described below is not set in stone, some steps need to be

[5]The device name /dev/rmt/tps1d6nrnsv.8500 is only an example a tape device name; the correct one should be determined for your system.

completed before others can be started. For example, it is not possible to run the signal preprocessing programs without the parameter file that is created by the mocomp preprocessor.

The majority of programs in the SAR processor are driven by a parameter file. The parameter file is constantly updated, modified and read from. A skeleton parameter file is required to start mocomp processing, and by the time all preprocessing has been completed, the parameter file will have been modified and will serve as the director for the actual compression program *atinsar1.4*.

The following steps show how to preprocess the mocomp and signal data. An attempt to describe the function of each step has been included.

1. **Locate glitches:** Glitches in the maid data need to be trimmed out. Glitches are irregularities in the data that can occur due to hardware failure. For example, the maid recording system may temporarily freeze and not record data for a few pulses. A glitch may also be caused by sudden aircraft motion such that recorded data cannot be smoothed properly. If not corrected for, glitches affect the smoothing and interpolation procedures. Usually, glitches are found at the beginning and at the end of the recorded maid data. The user is tasked with locating the glitches. The following procedure outlines one glitch detection scheme:
    a. **Load the maid file:** At the MATLAB prompt type: *load l8p9maid.mat*,
    b. **Plot acmd:** By typing *whos*, a list of all vectors in MATLAB memory is displayed. Plot the vector **acmd** and examine for glitches. In the diagram, figure 2, a glitch appears around sample 18000. Commonly, there will be a glitch at the start and at the end of a maid data file.

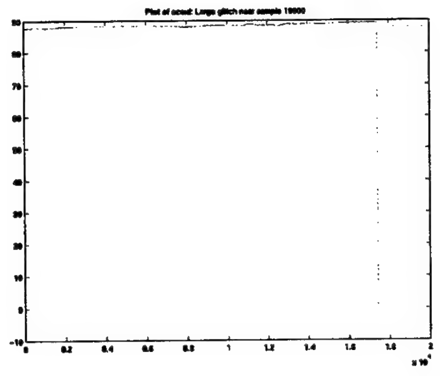


Figure 2: *Plot of acmd: A glitch appears around sample 18000*

c. **Plot derivative acmd:** Glitches may be more easily detected by plotting *diff(***acmd***)*. Effectively, the *diff* command causes the derivative of the vector to be calculated. Figure 3 shows the derivative of figure 2.

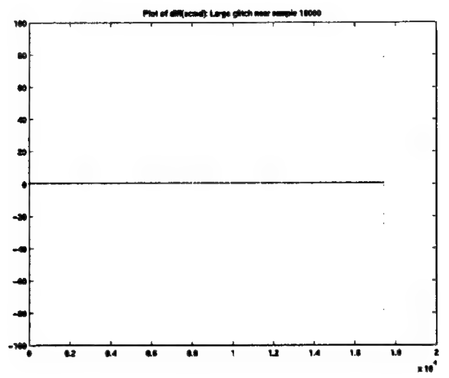


Figure 3: *Plot of diff(acmd): A glitch appears around sample 18000*

d. **Determine start and stop indexes for trimming:** From the two plots, the start sample and end sample of a clean (glitch free) set of maid data can be determined. These should be recorded for use in the next step.

e. **Find Corresponding start and stop in timeroll vector:** There are a number of vectors recorded by the maid system. Some parameters, however, are measured at different frequencies from others, which can be easily recognized by noting the sizes of the vectors in the MATLAB files. Parameters that are measured at higher frequencies will have more elements in their corresponding vectors than parameters that are measured at lower frequencies. Thus, it becomes an issue to translate the start and stop samples from one vector into the start and stop samples in another. If one vector is trimmed, others should be appropriately trimmed as well. The start sample and end sample for trimming for one vector can be related to the time that the sample was taken. Then, through the time reference, the start and stop samples for other vectors can be determined. For example, say that for **acmd** the start sample was determined to be 1788, then one could find the corresponding start sample in **timeroll** by typing:

```
t=timeacmd(1788);
pos=find(timeroll>t);
starttimeroll=pos(1)
```

Since different vectors are sampled at different frequencies, the frequency of the vector **timeroll** has been chosen to be the one at which times for trimming should be determined. When the start and stop time for the vectors in table 9 have been determined, the latest start time, and the earliest stop time should be used for trimming. The index of the latest start time and earliest stop time must be determined for a sampling frequency as used for **timeroll.**

MATLAB Vector	**Function**
acmd	Main antenna azimuth command
apos	Main antenna azimuth angle
desired_track_angle	Desired aircraft track angle
epos	Main antenna elevation angle
ppos	Main antenna pitch angle
ipos	Insar azimuth depression angle
ivspeed	Aircraft vertical speed
maid_f_gndspd	Groundspeed
maid_f_heding	Aircraft heading
maid_f_pitch	Aircraft pitch
maid_f_roll	Aircraft Roll
maid_f_trkang	Aircraft track angle

Table 9: *Vectors that should be inspected for glitches in the maid file*

2. **Trim glitches:** Glitch trimming can be done with the MATLAB program *startATI*. The start and stop samples for the **timeroll** vector must be known. The program is run by typing *startATI* at the command prompt in MATLAB and then selecting the first option for maid file glitch trimming. The output file should ideally be named differently from the input file so that the input file is not overwritten. For example, input file = l8p9maid.mat, output file = l8p9maidtrim.mat. An error may occur at this point if the latest start time and the earliest stop time of all of the vectors listed in table 9 have not been properly determined. The most probable cause of such a mistake is to assume that the termination time for recording for all vectors is consistent.[6]

3. **Convert to 64Hz:** The next step involves splining all vectors up to the same 64Hz frequency. As mentioned above, the recording frequencies for the different vectors is not consistent. Before use, all vectors need to be splined up to the same recording frequency (an interpolation for those vectors recorded at lower frequencies than 64Hz). The up-sampling operation is part of the *startATI* program. After the trim operation is finished, the user is asked whether or not to continue with the up-sampling operation. The output file from the operation could conveniently be named l8p9maid64.mat.

4. **Create MATLAB file of ancillary parameters:** Along with the maid data and GPS data, ancillary data collected for each pulse is generated by the radar system. The data must first be converted into MATLAB format. This is done with the program *ancmat2*, a shell program:

   ancmat2

   The output file from the program *ancmat2* is, unfortunately, not correctly named. It will have the extension .ancmat instead of .mat. The file should be renamed, *e.g. mv lxpxCa.ancmat lxpxCa.mat.*

   All four channels need to be unpacked, so the program needs to be run four times. The program is interactive. All available information should be converted into MATLAB format. Thus, all records should be dumped, from the first to the last record (the last record number will be provided by the program). All menu items should be chosen by entering 29 to toggle all. By then entering 0, the program will complete.

5. **Create parameter file:** Perhaps the most user intensive step is the creation of a parameter file. The parameter file is a simple text file defined by label and corresponding value on the same line. Each label/value pair is on its own line. There is a space between the label and value fields, but there is no character placement restriction. Since the software determines values by

[6]An absence of data constitutes a glitch.

looking for anything on the line after the first blank space, label entries must be such that they do not contain any spaces.

There are quite a few parameters that need to be entered by the user. As with the previous two operations, *startATI* provides an ideal tool to complete the operation. There is a third step included in *startATI* which prompts the user for all required parameters. The following tables (10, 11) list all parameters that the user will be prompted for, as well as default or suggested values.

There are four things worth mentioning about the parameter file:

- First of all, the suggested values are consistent with data that was collected in Nadir mode for a particular flight geometry and setup. The suggested values should all be verified against the system setup for the dataset that the user is processing. Oftentimes, there will be differences. For example, the range offsets between channel Ca and the others should really be measured, since it is likely to change from flight to flight.
- Secondly, some of the entries to be made into the parameter file will be the names of files that do not yet exist. Subsequent programs will read the parameter file and use information therein to name their output files.
- Thirdly, there should always be a terminating slash at the end of the data directory entry parameter value.
- Finally, not all of the information entered into the parameter file is used. Redundant parameters are represented by the "-" character in the suggested value column of tables 10 and 11 and/or it is explicitly stated that the parameter is not used.

For the rest of this guide, the parameter file will be referred to as lxpx.prm.

Parameter Name	Parameter Name	Suggested Value
flight_date	Flight date	
acquisition_year	Acquisition year	
julian_date	Day number of year	
line	Line of data	
pass	Pass of data	
data_directory	Directory containing raw data files	directory/
hddt_id_C	HDDT tape number for C channel	-
hddt_id_X	HHDT tape number for X channel	-
Ca_sig_tape_id	The signature id for channel Ca	lxpxCa
Cb_sig_tape_id	The signature id for channel Cb	lxpxCb
Xa_sig_tape_id	The signature id for channel Xa	lxpxXa
Xb_sig_tape_id	The signature id for channel Xb	lxpxXb
maid_tape_id	The name of the maid tape id	-
maid_file	The name of the maid file	lxpxmaid64.mat
moc_file	The name of the moc file	lxpxmoc.mat
prf_file	The name of the prf file	lxpxprf.mat
raw_data_Ca	The name of the Ca raw header file	lxpxCa.hdr
raw_data_Cb	The name of the Cb raw header file	lxpxCb.hdr
raw_data_Xa	The name of the Xa raw header file	lxpxXa.hdr
raw_data_Xb	The name of the Xb raw header file	lxpxXb.hdr
Ca_sigfile	The name of the Ca ancillary file	lxpxCa.mat
Cb_sigfile	The name of the Cb ancillary file	lxpxCb.mat
Xa_sigfile	The name of the Xa ancillary file	lxpxXa.mat
Xb_sigfile	The name of the Xb ancillary file	lxpxXb.mat
sig_time_file	The name of the signal/time file	lxpxsigtime.mat
image_file	The name of the interferogram file	lxpximg.mat
used_gps	Flag for GPS used or not	y
ivspeed	Flag for using ivspeed or not	1
main_ant_disabled	Flag for main antenna disabled	1
prfoverv	The prf over v value (in ancillary data)	2.57
moc_sar_delay	The mocomp recording delay	-0.23
gps_file	The name of the GPS file	lxpxgps.mat
gps_moc_delay	The GPS recording delay	0
insar_dep_angle	The insar antenna depression angle	30
insar_azi_angle	The insar azimuth angle	2.0

***Table 10:** Table 1 of input parameters*

Parameter Name	Parameter Name	Suggested Value
first_range_line	The first pulse to process	1
last_range_line	The last pulse to process	last pulse
near_edge_pixel	The first range gate to process	1
far_edge_pixel	The last range gate to process	4096
range_gate_delay	The range gate delay (ancillary data)	read from log
saw_delay	The surface acoustic wave delay	13.2737
nadir_pixel	The first nadir pixel detected	0
az_fracpix_offset_ba	The fractional offset between antennas	0.5911
delta_n_refgen		0
range_pix_offset_Cb	Range pixel offsets for Ca $\rightarrow$ Cb	0
range_pix_offset_Xa	Range pixel offset for Ca $\rightarrow$ Xa	-0.999
range_pix_offset_Xb	Range pixel offset for Ca $\rightarrow$ Xb	-0.999
processing_bw	The half power beamwidth (azimuth)	3.0
start_pct_bw	The start percent beam to process	0.0
stop_pct_bw	The stop percent of beam to process	100.0
fft_optimize	Optimize data chunk size for fft flag	n
smooth_factor	The smoothing factor	20
overlap_factor	The overlap_factor	2
veloc_file	The post processing velocity file (not used)	-
theta_file	(Not used)	-
corrected_image_file	The corrected image (not used)	-
phscor_file	The phase correction file	lxpxphscor.mat
ati_baseline	The antenna phase center separation	0.46
filter_smooth	smooth filter (not used)	0
gaussian_weight	Gaussian weight (not used)	0
veloc_filt	velocity filter (not used)	0
invalid_theta_flag	(not used)	0
invalid_image_flag	(not used)	0
invalid_velocity_flag	(not used)	0
utm_imgfile	(not used)	-
utm_velocityfile	(not used)	-
utm_zone	(not used)	0
utm_min_easting	(not used)	0
utm_min_northing	(not used)	0
utm_max_easting	(not used)	0
utm_max_northing	(not used)	0
utm_east_spacing	(not used)	0
utm_north_spacing	(not used)	0

Table 11: *Table 2 of input parameters*

6. **Combine GPS and maid data:** The GPS data and maid data need to be combined into one file, and to be splined up to the same sampling frequency. The MATLAB program *gpspro* accomplishes this task. The module is run by typing

   gpspro 'lxpx.prm';

   The program reads the parameter file created in the previous step. It extracts the name of the GPS file and the name of the maid file, opens them, and combines their data into the maid file. Already, the convenience of the parameter file becomes apparent. Two graphs are printed out at the termination of the program. They illustrate the degree of match between the GPS data and the maid data. They can be printed out and saved as part of the processing documentation.

7. **Create motion compensation vectors:** The processing program will need to know the correction offsets to compensate for motion errors. These corrections are calculated from the GPS and maid data and stored in MATLAB vectors by the program *atimocnew*. The syntax of the command is:

   atimocnew 'lxpx.prm';

   Again, all that is required is the parameter file name. This program writes data to the file specified as the .moc file in the parameter file.

8. **Test time differences:** The time differences between the four data streams (channels) needs to be established and tested. This is done with the MATLAB program *atitimetestnew*. The program is run by typing the following on the MATLAB command line:

   atitimetestnew 'lxpx.prm';

   The program will output information concerning the four data streams and raise any warnings stemming from timing problems. The most probable cause of a timing error would be rooted in the signal extraction from the SONY helical scan tape onto the exabyte tape.

9. **Combine ancillary parameters, GPS and motion compensation data:** The ancillary data needs to be splined up to the GPS and maid data. The program *atiprf* completes this task. The program is run by typing:

   atiprf 'lxpx.prm';

   at the MATLAB command line.

10. **Data Quality Control:** A test and fix of potential errors is done by running *qcnati*. The program is an executable that is run from the shell:

qcnati lxpx.prm

It is not a fast program, and for 40000 pulses on an SGI Octane, should complete in around a half and hour.

11. **Equalize channels:** The first and second moments of the real and imaginary parts of each of the channels needs to be calculated. Later on, during the processing stage, the information is used to balance the channels so that they appear all to be measured with the same scale. The program responsible for calculating the first and second moments of each channel is called *meandati*. It is an executable, and is run by typing:

    meandati lxpx.prm

    at the shell prompt. The program completes fairly quickly.

That's it! The main part of the user's work has been done. Although lengthy in duration (running time), all that remains is to run the compression program, which does not require any user interaction, and then to calibrate the data.

4.7 Three: How to run atinsar

The main processing engine is called *atinsar1.4*. There are various versions of the program, the most current stable version being *atinsar1.4*. The program is run from the command line. Documentation about the program and how it runs can be found in the delivered product section.

The processor will take about 2 hours to process 40,000 pulses of data on an SGI Octane computer. The output from the program is the MATLAB image_file specified by the parameter file.

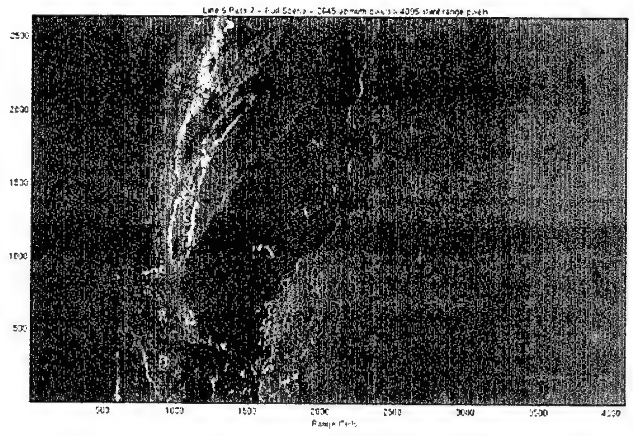
After processing, it is important to check that the coherence between the two channels is high enough for moving target detection. If the program miscalculates the height of the radar because a non-existent nadir pixel has been used, then the coherence will be an undefined quantity. If the coherence is NaN, then the data should be reprocessed with the correct value for the first nadir pixel.

4.8 Four: How to phase calibrate the data

The final step in data processing is calibration. This is a fairly straight forward step. The data is not actually calibrated, but the phase calibration is determined for later use. The phase correction file as specified in the parameter file is created by the program *landphscal*.

The only tricky point to keep in mind is that phase calibration is dependent on a sample of signal data over which the phase is "well behaved". Any sections of water that appear in the image need to be removed for phase calibration as water does not provide a good reference target for phase calibration.

A reasonably sized section (400 pulses) of imagery that stretches across the swath will be required. The section can be a patchwork of sections of different pulses the only caveat being that the whole swath is covered. For example, if in the output interferogram, there is some section where there is land in the far-range and water in the near-range, and if there is also a section (elsewhere) where there is water in the far-range and land in the near-range, then land (from near and far) can be cropped and reconnected so that all sections of the swath are represented by land.



Line 5 Pass 2 - Full Scene - 2645 azimuth pixels x 4095 slant range pixels
2500
2000
1500
1000
500
500
1000
1500
2000
2500
3000
3500
4000
Range Pixels

Figure 4: A SAR image containing water patches

Cropping and patching together can easily be done in MATLAB. The resulting data should be saved in a file such as lxpxlandcal.mat. Suppose a situation exists as illustrated in figure 4 and we wish to extract a section of data from the bottom, near-swath part of the image and reconnect it to a section of imagery from the top, far-swath part of imagery. The following would suffice:

```
load lxpximg.mat
imagesc(abs(ougs),[0,15]); colormap(gray)
subougs=ougs(2001:2400,1:1800);
subougs(1:400,1801:4096)=ougs(1:400,1801:4096);
ougs=subougs;
save lxpxlandcal.mat;
```

landcal('lxpx.prm','lxpxlandcal.mat','0.01');[7]

A land calibration file of 400 samples of land data would be created, and then a phase correction file will be created from that data. The phase correction file will contain two vectors: one, the incidence angle to the range cell, and the other, the phase correction for the interferogram.

The data has now been processed and is ready for analysis via post processing software.

[7]The '0.01' is the minimum magnitude signal that will be included in an average for the phase estimation. Anything below it is considered noise.

5. Changes made to the program

The name of the program received from CCRS was *atinsarnew*. During porting, modification and enhancement, the name of the program was changed to *atinsar1.0*, then to *atinsar1.1* and so on, up until the current version *atinsar1.4*.

The following is from a README file found with the software. It was written by John Campbell. The notes referred to are the notes that John himself compiled (found later in this document).

> The ATINSAR software is an adaptation from the InSAR software. Initial work on the software was accomplished by Marco van der Kooij. Most recent work was accomplished by John Campbell on an SGI with old IRIX software. See the InSAR notes for some of the necessary changes to the C-coding.
>
> MATLAB .mex files were also written for an older version of MATLAB and need to be updated.
>
> TAPDMP and SIGDMP are programs of John Wolfe and have to be obtained from Paris Vachon.
>
> PRISM or the equivalent is required for processing the GPS data.
>
> GPSMAT assumes PRISM formated output.
>
> IMGLIB, GPSMAT, INSARMAID, ANCMAT, GPSPRO, STCCAL and other subroutines are common to atiSAR and InSAR and are located in the directory "../common".

The above note indicates that the program was originally compiled for an old version of IRIX (something prior to IRIX 6.5) and that changes had to made to the code to allow the program to be compiled.

5.1 Changes made to compile the processor

The first task was to compile the program. Because of backwards compatibility, the only compilation issues had to do with improper path specifications. Specifically, #include statements had to be adjusted to indicate the correct locations of files. The following notes were written by Mike Robson who first compiled the program at DREO:

1. utilsub - 2 object files
2. ancmat

- 3 object files to start, only 2 come out of make
- missing imglib.o
- it should use the one in ../utilsub

3. atinsarnew - 1 executable file, same size as the old file
4. gpsmat 1 executable file and one object file, both same size as old versions
5. insarmaid 1 executable and 1 object file, both same size as the old files
6. meandati
   - meandati.c had 3 include statements at the end with the wrong path names ("." instead of "..")
   - fixed it up
   - object file and executable both smaller than the original
7. qcnati
   - same problems on all accounts as meandati
8. sigdmp
   - make Makefile : all OK
   - make dispanc.mak
   - dispanc.c had wrong directory for insar_prot.h
   - dispanc.o is bigger than before
   - dispancrec2.c also had wrong directory for insar_prot.h
   - different file sizes than previous versions on dispancrec2.o and dispanc
   - make fixyear.mak
   - fixyear.c had wrong directory for insar_prot.h
   - dbltorel.c had wrong directory for insar_prot.h
   - dbltorel.c error in argument in strcpy line 91
9. complain on
   - abstorel(strcpy(...),year,julianday,reltime)
     abstorel template is abstorel(char*,long*,long*,long*)
   - strcpy(abstime,dbltoabs(dbltime))
     the system thinks strcpy returns int not char*
   - also in fixyear.mak, -lc_s caused an error so I commented it out

5.2 Changes made to take advantage of the SGI platform capabilities

Supplied SGI libraries provide system specific optimized methods for accomplishing tasks that might otherwise take up more time. One of the time burdensome tasks is the convolution of the azimuth reference function with the raw data. The SGI library libcomplib.sgimath provides an efficient mechanism for computing Fast Fourier Transforms, a key step in the convolution operation. In addition, the SGI libraries provide efficient methods for manipulating arrays of complex data.

The original method for computing the FFT's is described in such books as [1]. This and similar methods, while capable, are not tuned to the architecture of the SGI machines.

The changes made to convert to the SGI FFT routines involve the following:

1. The call to convlv2 was changed to a call to a new function called convlv_sgi with additional arguments. The structure of convlv2 was as follows:

   void convlvc2(float *in1,float *in2,float *out1,float *out2,int n,float *respns,int m).
   This was changed to

   void convlv_sgi(float *in1,float *in2,int n,float *respns,int m,float *workspace)

   In convlvc2 and convlv_sgi, the first two arguments are the same, but in convlv_sgi the input arrays are themselves modified and returned, thus eliminating the need for additional arrays for output (out1, and out2 arrays are no longer a part of the function call). In addition, there is a new variable passed to the function called workspace, an array that is required by the SGI FFT routines.

2. The SGI FFT routines use a workspace array as a catalyst to the FFT routine. More information can be found on the SGI FFT routines in [2]. The workspace variable is declared in the code and initialized with SGI routines. The name of the initialization function is cfft1di.

3. The new function convlv_sgi was written to do the FFT and convolution operation. Inside the new function definition, various new SGI routines are implemented, such as

   - cfft1d: performs FFT
   - cprod1d: does multiplication of complex arrays

- cscal1d: does a scalar multiplication of a complex array

4. Array indexing started at 1 in the old FFT routines, but for the SGI routines, array indexing starts at 0.

5.3 Changes made that enable the whole dataset to be processed

Initially, only 2048 range lines could be processed at a time. This limit has been extended so that now the only limitations are with available time and disk space. There still remains, hard-coded into the function opesuni.c, the limitation that the number of range lines must be divisible by 512. Since it is unlikely that the data format will ever again be presented in UNIDSK format, the use of opensuni.c is questionable.

The method by which the extended number of range lines was implemented was simply to change the following code segment:

```
delresCb = fvector(2056);
delresXa = fvector(2056);
delresXb = fvector(2056);
for (j=0; j<2056; j++)
```

to

```
delresCb = fvector(p2+SPLINE_BUFFER);
delresXa = fvector(p2+SPLINE_BUFFER);
delresXb = fvector(p2+SPLINE_BUFFER);
for (j=0; j<p2+SPLINE_BUFFER; j++)
```

where p2 is the number of the last range line to be read in, and SPLINE_BUFFER is 8. If the first range line is not 1, then the above can still be improved (and will be) by using (p2-p1+1) instead of p2.

5.4 Changes made to enhance moving targets

The reference function used for azimuth convolution was modified to take into account the phase structure of a target with motion. The section on the modified reference function presents a derivation of the new reference function.

5.5 Changes stemming from MATLAB 5 Considerations

Most of the MATLAB functions were written for version 4. Although backwards compatibility existed, a special switch had to be specified when starting MATLAB 5 so that data formats would not conflict. For example, running the program *insarmaid* on a data set without the *-v4* option specified when MATLAB was started meant that the program *newstartati* could not be run. That is, if the MATLAB session for *newstartati* was started with the *-v4* switch, it could not read the data produced by *insarmaid* (when the switch was not specified). If the session for *newstartati* was started without the *-v4* switch, then the .mex files upon which it depends could not be found and the program would again not complete. The .mex files could not be found because they were compiled for version 4.

Another problem arose after MATLAB version 5 was released that had to do with reading matrix files from C-programs. Because the motion compensation data was written in MATLAB format, it was necessary to read the MATLAB matrix format from a C-interface. Unfortunately, after version 4, MATLAB decided to make their file format proprietary, so that the only way to read MATLAB files from the C-interface was through libraries provided by MATHWORKS. Because of backwards compatibility, writing files from C to MATLAB format was not a problem; however, it became an issue to read the MATLAB files from the C-programs if the file format was of version 5.

Both of the above problems stemmed from a need to keep apace of changes and improvements to MATLAB. It was decided that it was preferable to upgrade the versions of the MATLAB software and C-interface instead of seeking new mechanisms of linking the old versions of both the software and file format with the new. Thus, changes were made that upgraded the MATLAB dependence of the program to version 5. There were two main thrusts to the upgrade to version 5, and these had to do with the two problems described above.

First of all, the .mex files, compiled from C-programs, were recompiled with the new version of MATLAB. One of the implications of this was that the C-programs were compiled in 64-bit mode. Care was taken to make sure that there would be no conflicts between data type length. In 64-bit mode, a variable of type long is 8-bytes in length instead of 4-bytes in length. All structures that contained data of type long were examined and longs were changed to ints whenever it was determined that the long could be contaminated if data was being assigned to that structure (*i.e.* by changing long to int, the structure sizeof does not change in 64-bit mode). Compilation of the C-programs is described in section 3.3.

The second upgrade task was a little more involved, as it had to do with directing the program to use supplied MATLAB libraries from the C-interface. Since it was decided that all the C-programs be compiled in 64-bit mode, the same problem with data type length was fixed as above, but an additional problem involved removing the old MATLAB read routines and applying the supplied routines in such a way that they seamlessly fit into the program. The old *imglib* routines were completely removed, and a wrapper function called *getmatlab_vecd* that relies upon the supplied MATLAB libraries was written and included in a new MATLAB I/O library. In addition to the aforementioned wrapper function, the write function for MATLAB file format was included in the new library as well. The write function put into the MATLAB I/O library was constructed from some of the code that already existed at the end of the main program source (with modification). The read function was created from scratch.

5.6 Changes made to the operational procedure of the program

Changes to the operation mode of the program were instituted. One of the major changes along these lines was the inclusion of command line arguments. Command line arguments provide the opportunity to pass parameters that might be used for defining the matched filter, or that might be used to flag the program to output fore and aft channel images, as well as the interferogram. The list of available command line arguments is given in table 1.

5.7 Changes made to enable other detection schemes

ATI methods are only one way of examining the information contained in two independent channels of information. Displaced Phase Centre Analysis (DPCA) is also a well explored, scheme for moving target detection [3].

The atinsar processor was designed to output an interferogram at the end of the processing step. Now, with the option to output both the fore and aft channels separately, it is possible to ingest the separate channels into any post-processing program and perform different types of analysis.

In June 2000, the option of outputting the fore and aft channels existed, however, it was only possible by changing the value of a hard-coded flag inside the program. It would have been necessary to change the value and recompile the program every time that the output of fore and aft channels was desired. A better method was developed where a command line option, *-f*, directed the processor to output the separate channels. The names of the output channels *fore.mat* and *aft.mat*, are hard-coded into the program. The default is to not output the separate channels.

One thing to keep in mind when the separate channel mode of the processor is run, is that the processor will down-sample both the interferogram and the separate channels. The interferogram, however, is first calculated and then detected (by down-sampling) while the separate channels are simply detected. Therefore, if subsequent programs ingest the separate channel information hoping to reproduce the interferogram, a problem will arise in that the channels are detected first and then the interferogram is created; the reverse occurs inside the processor. It is possible to output Single Look Complex (SLC) data in the fore and aft channels, *i.e.* no down-sampling. It is probably preferable to work with the SLC data and then do detection, rather than to work with the detected data.

5.8 Changes made to the design of the code

To comply with common coding practices, the design of the code was quite radically changed. There were a number of reasons for altering the code structure, the main probably being that it facilitates future upgrade. Other reasons are that the code is easier to read, that separate sections can be worked on independently because of modularity, and that portability in terms of compiling can be done more easily.

The following is a list of elements of the code that were undesirable:

1. The inclusion of function source code that is used in other programs via #include directives,

2. The absolute path specification in header and source code #include directives,

3. The conglomeration of many unrelated functions into one file *atinsarnew.c*,

4. The fact that all function declarations were done in one file *insar_prot.h*.

All of the above concerns were handled in the following manner respectively:

1. Related functions were compiled and linked into dynamic libraries to which the main program links at run time. These functions include all of the following:

function name	**library into which it was put**	**Source file**
initmatfile	libmatio.so	getmatlab_vector.c
matwriterow	libmatio.so	getmatlab_vector.c
getmatlab_vector	libmatio.so	getmatlab_vector.c
getmatlab_vecd	libmatio.so	getmatlab_vector.c

***Table 12:** MATLAB I/O library*

2. The absolute path structure was removed. All of the following header files were put into a common place:
   - matlab_vector.h,
   - argsubs.h,
   - chcksub2.h,
   - local_error.h,
   - vectors.h,

function name	library into which it was put	Source file
downgaus	libc_numerical.so	downgaus.c
convlv_sgi	libc_numerical.so	convlv_sgi.c
convlv2_sgi	libc_numerical.so	convlv_sgi.c
spline2	libc_numerical.so	spline2.c
free_local_fvector	libc_numerical.so	vectors.c
local_dvector	libc_numerical.so	vectors.c
local_fvector	libc_numerical.so	vectors.c
local_free_fmatrix	libc_numerical.so	vectors.c
local_ftmatrix	libc_numerical.so	vectors.c

Table 13: *Numeric function library*

- readsuni.h,
- downgaus.h,
- convlv_sgi.h.

The location of the header files is now passed to the compiler through compiler switches in the Makefiles for the program and the libraries. The following command is used to indicate the include file location

-I/header-file-location

3. The file *atinsarnew.c* was changed so that only the main function is defined. The other function definitions such as *downgaus* and *local_fvector* have been moved to separate files and are now compiled into separate libraries as described in item 1 above.

4. The main header file *insar_prot.h* was abandoned. Now, all files have a corresponding header file closely related to their function. The list of header files in item 2 above contains all of the function declarations that were initially in *insar_prot.h*.

strlow	libc_shared.so	chcksub2.c
strmfe	libc_shared.so	chcksub2.c
extchm	libc_shared.so	chcksub2.c
strupr	libc_shared.so	chcksub2.c
extchck	libc_shared.so	chcksub2.c
plchck	libc_shared.so	chcksub2.c
pow2chck	libc_shared.so	chcksub2.c
skip_junk	libc_shared.so	argsubs.c
end_junk	libc_shared.so	argsubs.c
iscomment	libc_shared.so	argsubs.c
check_key	libc_shared.so	argsubs.c
argScanLog	libc_shared.so	argsubs.c
argScan	libc_shared.so	argsubs.c
argWriteLog	libc_shared.so	argsubs.c
argWrite	libc_shared.so	argsubs.c
argAppendLog	libc_shared.so	argsubs.c
argAppend	libc_shared.so	argsubs.c
argOverwrtlog	libc_shared.so	argsubs.c
argOverwrt	libc_shared.so	argsubs.c
argExistLog	libc_shared.so	argsubs.c
argExist	libc_shared.so	argsubs.c
argWrite_intlog	libc_shared.so	argsubs.c
argWrite_floatlog	libc_shared.so	argsubs.c
argWrite_float	libc_shared.so	argsubs.c
argScan_intlog	libc_shared.so	argsubs.c
argScan_int	libc_shared.so	argsubs.c
argScan_floatlog	libc_shared.so	argsubs.c
argScan_float	libc_shared.so	argsubs.c
argScan_stringlog	libc_shared.so	argsubs.c
argScan_string	libc_shared.so	argsubs.c
argCopyfilelog	libc_shared.so	argsubs.c
argCopyfile	libc_shared.so	argsubs.c

Table 14: *Shared C Function Library*

5.9 Changes made to the design of the program

The main change made to program design was to compile the utility functions into separate dynamically linkable libraries. Both the 32-bit and 64-bit versions of these libraries were compiled. The three new libraries are detailed in the previous section.

5.10 Changes made regarding documentation

An effort was made to document the code. Included are a series of notes (next chapter). Also, comments were written directly in the code. Many notes compiled by John Campbell were used to compile the notes in the next chapter.

6. Notes on the processor

Many aspects of the processor were examined and notes describing many of the more poorly described sections have been compiled. Many of the following notes were compiled by John Campbell, as indicated by a header in the section.

Some of the following notes describe functions and subprograms, others describe some of the models upon which the code is/was based.

6.1 gpspro

CCRS Airborne Along Track Interferometric SAR Processing Software Revision 1.0 - As received from CCRS by DREO.

PROGRAM DESCRIPTION - GPSPRO.M
John W. M. Campbell, April 22, 1999

This program takes the MATLAB GPS file produced by gpsmat and merges it with the 64 Hz MAID data MATLAB file produced by startATI.

The parameter file is used to acquire the appropriate file names and the variable gps_moc_delay, which gives the time offset between the MAID time code and the GPS time code. For historical data sets, gps_moc_delay was normally some fraction of a second, but it should now normally be zero.

The calling sequence in MATLAB is thus:

gpspro(paramfile),

where paramfile is a MATLAB string array containing the name of the parameter file.

The program makes plots of some of the splined GPS data, and it once was used to allow for choosing a previously saved set of axes for making a hardcopy printout, but these options have been turned off for some reason.

Initially, the MAID data and GPS data are read in, and the timecode function is used to display the time extent of both files in a standard ASCII form.

Both MAID time (time) and GPS time (gpstc) are converted to times relative to the first MAID time sample (with the additional gps_moc_delay offset for GPS).

The next operation is to compute a GPS ground speed from the lat, long and elevation values. The comment that R is "average elevation for great circle" is bogus, and it would be more appropriate to have difftime = gpstc + diff(gpstc)/2.

Next the subroutine trackcal, a .mex file, is used to compute the track angle between each pair of lat,long points. The 100th point in the MAID track angle array is arbitrarily chosen to determine if the track is within 15 degrees of true north (in which case the MAID track and heading are both set to a range of -180 to +180 instead of 0 to 360). The same test is done using the 100th GPS track sample after the first MAID time stamp. Since these two conversions may have brought the GPS and MAID angles into different regimes, it does a final check and adds or subtracts 360 degrees from the GPS track if required.

The program now splines all of the gps variables to a 64 Hz sampling rate (not 40 Hz as it says in the code) using the cspline .mex file. The difftime variable is used for gps_vg and gps_track, because these were calculated from pairs of gps positions and are thus shifted by half a sample. It appears that they are shifted in the wrong way, however, since difftime is defined wrong (see above). gps_vg and gps_track are also smoothed using a 250 point low pass FFT filter (applied in the .mex file smooft). Why 256 was not chosen is unknown.

N.B. smooft, like all other programs in this suite, is using very old FFT routines written at CCRS. It would most likely benefit from use of the FFT routines specifically designed for SGI which are provided with the IRIX 64-bit compilers.

Next, the extra samples included in the spline to avoid edge effects are trimmed off and the Cartesian coordinates in the WGS84 system are computed from the GPS lat, long and elevation. The MAID track angle is also smoothed using smooft at this time. All the new variables are converted to row vectors and appended to the MAID MATLAB file using the .mex file append. For no obvious reason, time is appended as well, even though it originally came from this file and hasn't been changed.

N.B. Since the append .mex file is unavailable, the version from the polarimetry software has been substituted. It is supposed to be compiled using the header from Ian Neeson's code (./c_ian/argsubs.h), but after looking at the code, this subroutine will probably compile with the old argsubs.h file. If it does not, then the argsubs from the polarimetry software should be renamed (to avoid conflicts) and added to the InSAR software (with the appropriate change to the #include statement in append.c).

Finally, comparison plots of MAID track with GPS track and MAID groundspeed with GPS groundspeed are made.

6.2 meandati

CCRS Airborne Along Track Interferometric SAR Processing Software Revision 1.0 - As received from CCRS by DREO.

PROGRAM DESCRIPTION - GPSPRO.M
John W. M. Campbell, April 23, 1999

This program reads the signal data contained in the four edited .sig files output by qcnati (it actually uses the .hdr file names input during the third step of startATI as part of the parameter file to access them). The calling sequence is

meandati paramfile,

where paramfile is the name of the parameter file defined in startATI.

N.B. There are compiler macros (#define statements) in this program. Hopefully they will work with different compilers. It is strongly recommended that a test program be compiled to verify their performance.

The readsuni library (which is a stripped down holdover from the UNIDSK library) is used to open the file and get information on it. This isn't really necessary because the signal files are always John Wolfe style flat files, but it should work.

N.B. This does cause an error, however, if the number of pixels in the signal data files opened is not a multiple of 512.

This program is a bit smarter than qcnati. It opens all four signal data files and verifies that the pairs of files from each PCU have the same number of lines and pixels. It uses the max_cache parameter from the parameter file created by startATI to determine how many lines can be read at once. N.B. It appears to handle any possible remainder lines correctly!

The program simply computes the sum and sum of squares for the I and Q pixels in each line of the file. Then it divides by the number of pixels in the line. At the end of the disk ingest loops for each channel, it divides by the number of lines.

Finally, it writes the mean and mean of squares for I and Q for each channel to the parameter file. These values are later used to perform channel amplitude balancing when merging the four input files in ATINSARNEW.

6.3 qcnati

CCRS Airborne Along Track Interferometric SAR Processing Software
Revision 1.0 - As received from CCRS by DREO.

PROGRAM DESCRIPTION - GPSPRO.M
John W. M. Campbell, April 16, 1999

This program reads the signal data contained in the four .sig files produced by sigdmp (it actually uses the .hdr file names input during the third step of startATI as part of the parameter file to access them). The calling sequence is

qcnati paramfile,

where paramfile is the name of the parameter file defined in startATI.

N.B. There are compiler macros (#define statements) in this program. Hopefully they will work with different compilers. It is strongly recommended that a test program be compiled to verify their performance.

The readsuni library (which is a stripped down holdover from the UNIDSK library) is used to open the file and get information on it. This isn't really necessary because the signal files are always John Wolfe style flat files, but it should work.

N.B. This does cause an error, however, if the number of pixels in the signal data files opened is not a multiple of 512.

This is where the programs start to show the nightmare caused by all the different people who revised them. A lot of the lines of code in this and subsequent programs are irrelevant, or perform calculations that don't accomplish anything. Unfortunately, they probably made sense in the earliest versions of the code and nobody was willing to change the style, so things were just set to constants, or quantities were calculated and then ignored.

One by one, the program opens the signal data files and displays header information. It assumes the number of lines and pixels in Channel Ca will be exactly duplicated for the other three channels and attempts to allocate an array, cash, which is large enough to accommodate 512 lines * number of pixels per line in Channel Ca * 2 (for both I and Q). This array is then used to ingest 512 line chunks of the file for analysis. The program also calculates the number of loops, linloops, and whether or not there will be any remainder loop required using the lines and pixels for channel Ca.

After it has opened and displayed information for each file, the main loop of the program reads in the data in 512 line chunks.

N.B. It appears that the first time the program goes through the main body loop, if there is a remainder loop (ie. linflag = 1) then the program will set nlin = remlin, the number of remainder lines on the last loop of the for (jj=1; jj<=(linloops+linflag); jj++) statement. Then, when it comes back to this statement for the next channel on the for (ij=0; ij<4; ij++) statement, nlin will still be remlin, and the mean and standard deviation calculations will be in error. Also, even the first time the remainder loop is executed, the fread statement still attempts to read DCASHSIZE bytes, with unknown effects. This probably never showed up because we always used a power of 2 as the number of records to process, so remlin was zero.

For each 512 line tile from an individual signal file, the data is read into the array cash. For each range pixel and for each of I and Q bytes separately, the mean and standard deviation over the 512 lines are calculated.

The following errors are corrected:

1. Any variation by more than 7 standard deviations is changed using the nearest power of two which minimizes the difference between the pixel and its two nearest neighbour lines, provided that power of two is larger than 16.
2. If the pixel is saturated (+62 or -64), a warning is issued but no data is changed.
3. If the pixel is beyond the saturated value, either 64 or 128 is added or subtracted, as necessary, to bring it into the normal range.
4. If the pixel is odd, it is made even in such a way as to minimize the difference between it and its two nearest neighbours in adjacent azimuth samples (lines).

The corrected data is written back to the input file.

This methodology is correct if we assume that the data is sampled at 6 bits and written into bits 2 to 7 of an 8 bit word, and that all errors are due to individual bit errors in the data stream. This should be correct.

6.4 ancmat

CCRS Airborne Along Track Interferometric SAR Processing Software Revision 1.0 - As received from CCRS by DREO.

PROGRAM DESCRIPTION - ANCMAT.C
John W. M. Campbell, April 15, 1999

This program uses the toggle_menu subroutine to solicit a user supplied set of ancillary parameters from an ancillary file created by sigdmp. It copies these parameters from a user selected subset of the records in the file into a MATLAB 4.2 format file.

The user should usually select all of the available records, but the only parameters to be copied are TIME and HDDTREC.

N.B. This program also uses imglib to access MATLAB.

6.5 atitimetestnew

CCRS Airborne Along Track Interferometric SAR Processing Software Revision 1.0 - As received from CCRS by DREO.

PROGRAM DESCRIPTION - ANCMAT.C
John W. M. Campbell, May 18, 1999

This program examines the hddtrec and time variables copied to MATLAB files from the HDDT ancillary data records for each of the four channels, PCU1-A, PCU1-B, PCU2-A and PCU2-B. It determines the correct ordering of the four channels of data for the along track interferometry case.

The calling sequence in MATLAB is:

atitimetestnew(paramfile),

where paramfile is a MATLAB string array containing the name of the parameter file.

First, the files defined by the arguments Ca_sigfile, Cb_sigfile, Xa_sigfile and Xb_sigfile in the parameter file are loaded into MATLAB one by one. The time and hddtrec arrays from each one are copied to catime, carec, cbtime, cbrec, etc.

Since channel A and B from the same PCU should always have the same record numbers (and these should increment in twos), we verify that diff(??rec) is equal to 2 everywhere and that the total number of records covered by ?arec and ?brec are the same.

The parameter az_offset_CbCa is then set to carec(1)-cbrec(1) and written to the parameter file. If it is zero, then the fore and aft data from PCU1 are perfectly aligned. If it is negative, then the fore data start at a lower hddtrec number (*i.e.* an earlier pulse) and, if it is positive, the aft data start at a lower hddtrec number. This can happen because the stripping of the four ancillary files may have been based on timecode and not HDDT Record Number. A similar process occurs for parameter az_offset_XbXa (set to xarec(1) -xbrec(1)).

Next, the timecodes are multiplied by 1000 and converted to integers to do the final registration. These variables are stored as ??time2. The program then verifies that ?atime and ?btime are identical everywhere by checking that sum(abs(?atime-?btime))=0. It also makes sure that diff(??time2)<6 everywhere (*i.e.* there are no sudden jumps in the time codes of more than 5 milliseconds). It actually does this calculation for all four channels, but it really only needs to check catime2 and xatime2, since it has already verified that ?atime and ?btime are identical.

The time code ordering allows either the C (PCU1) or X (PCU2) channel to come first for both antennas (A and B channels have already been checked to verify they have the same timecodes, but it does not allow for the time codes to be different by more than one sample. Thus, if catime(1) < xatime(1) AND catime(2) > xatime(1), the ordering for both the fore and aft antennas is C1, X1, C2, X2, C3, X3, ..., CN, XN.

If catime(1) >= xatime(1) AND xatime(2) > catime(1), the ordering for both antennas is X1, C1, X2, C2, ..., XN, CN.

Any other logic combination on these timecode tests is considered an error in the time code records. Since this is done, the former variables cstarta, cstartb, xstarta, xstartb and fore_to_aft_offset are irrelevant and can be set to 1 (?start?) and 0 (fore_to_aft_offset).

Finally, catime and xatime are merged in the correct order into foretimes, the combined time stamp record for the fore antenna, and cbtime and xbtime are merged into afttimes.

The parameter file parameter hddtrec_raw is written using carec(1), since PCU1A is considered the reference, and the array hddtrec is set to carec. Then, the output variables and arrays are saved to the file defined by the parameter sig_time_file.

6.6 atiprf

CCRS Airborne Along Track Interferometric SAR Processing Software Revision 1.0 - As received from CCRS by DREO.

PROGRAM DESCRIPTION - ANCMAT.C
John W. M. Campbell, May 19, 1999

This program takes a subset of the MAID and GPS combined data, which has produced the values in the moc_file, and splines it up to the rate of the interleaved time stamp in sig_time_file (produced by ATITIMETESTNEW). The subset is defined by the time extent of the data in sig_time_file, after accounting for a fixed delay of 64 records.

The calling sequence in MATLAB is:

atiprf(paramfile),

where paramfile is a MATLAB string array containing the name of the parameter file.

First, the program reads in the output of ATITIMETESTNEW from the sig_time_file and the MAID and GPS data from the moc_file produced by ATIMOC. It uses the interleaved times for the fore antenna, foretimes, as the master time record, and uses the .mex file sigtime to condition the time codes so that there are no adjacent records with the same values. This was probably necessary because at some point the time code was not recorded with sufficient precision. It then shifts the data by 64 records to the right, then backfills the first 64 samples by subtracting integer multiples of 1/PRF from the 65th time record.

N.B. This procedure is designed to accommodate a 64 PRF delay between when the data is received and when it is time stamped. This is assumed to be correct but is not proven.

Next, the program extracts a subset of the MAID and GPS data for +0.5s and -0.5s beyond the time extent of the 64 PRF shifted foretimes array stime. This is defined by the indices n1 and n2, which give the first and last records in all the MAID and GPS variables to use.

The parameter azimuth_pixel_spacing is then written to the parameter file. It is the average of the spacing over the indices n1 to n2.

Next, the subset of MAID times is referenced to stime(1) and the factor moc_sar_delay is subtracted so that x=time-stime(1)-moc_sar_delay.

N.B. Note that this procedure assumes that there is a delay of moc_sar_delay seconds between the time for which the MAID and GPS records are valid and the time at which the MAID time stamp was acquired. The GPS data has already been shifted by gps_moc_delay in the program gpspro. Both of these parameters are user inputs!

The cspline .mex file is then used to spline a selection of the parameters from the time coordinate defined by x to the one defined by xi=stime-stime(1). This list includes: delv_main, delv_ati1, delv_ati2, delh_main, delh_ati1, delh_ati2, prf, mocomtrack, delay_ati, gps_vg and if GPS is available, gps_elev, Xant, Yant, Zant.

These variables, along with stime (now renamed to time) and hddtrec (with 0.0000001 added to make the values non-integer (reason unknown)), are saved in the output file prf_file.

The parameter delv_ins2main is also written to the parameter file. It is simply the mean of the splined delv_main.

6.7 atimoc

CCRS Airborne Along Track Interferometric SAR Processing Software Revision 1.0 - As received from CCRS by DREO.

PROGRAM DESCRIPTION - ANCMAT.C
John W. M. Campbell, April 26, 1999

This program uses the GPS data in the file produced by gpspro to remove long term biases from the MAID data in the same file.

The parameter file is used to acquire the appropriate file names and many parameter values.

The calling sequence in MATLAB is

atimoc(paramfile),

where paramfile is a MATLAB string array containing the name of the parameter file.

First, there is some initial reading of parameters from paramfile and this time the tracks are converted to the range -180 to +180, so long as the 100th sample in desired_track_angle (not maid_f_trkang) is within 15 degrees of true North.

The main antenna elevation array, epos, is replaced by its mean value and this is written to the parameter file. Similarly, the mean of the main antenna pitch array is stored in mppos and written to the parameter file, while the actual pitch array is cleared.

Twenty Individual processing steps follow.

1. A straight line is fitted to gps_track (or maid_f_trkang, if GPS unavailable) and stored in mocomtrack.
2. The maid_f_trkang is filtered with a 50th order low pass FIR filter using a Hamming window, and stored as ins_track.
3. The difference between the filtered maid_f_trkang and gps_track is fitted to a straight line and subtracted from the filtered maid_f_trkang to get the true unbiased track, actual_track. The difference between actual_track and the straight line fit to gps_track from step 1 is stored in mocomang. If no GPS is available, then the filtered maid_f_trkang is stored in track and the difference between that and the linear fit to maid_f_trkang is mocomang.
4. The maid_f_gndspd is filtered and stored as vgfilt.
5. The difference between the filtered maid_f_gndspd and gps_vg is fitted to a straight line and subtracted from the filtered maid_f_gndspd to get the unbiased ground speed, actual_vg. The actual prf is then prf=actual_vg*prfoverv. Since the real PRF is slaved to maid_f_gndspd, this seems wrong. If no GPS is available, prf is nominally set to the filtered maid_f_gndspd times prfoverv.
6. The inter pixel spacing in azimuth is spacing=actual_vg/(vgfilt*prfoverv). If no GPS, spacing=1/prfoverv. This seems correct.
7. Since mocomang is the sample by sample difference between a nominal linear track and the track measured by the MAID system (after correction for any linear bias offsets), it can be thought of as a momentary deviation in the aircraft trajectory from nominal track. This is separated into along and across track velocity components, which are offsets to the aircraft's nominal velocity along its track. These are calculated using mocomang and actual_vg (or vgfilt if no GPS available).
8. Since insarmaid and startATI don't deal with the vertical acceleration data, used_ivspeed should be 1, so this step will simply filter ivspeed and convert it from feet/minute to meters/second, saving the result in vel_vert.
9. The across track velocity, crosstrackvel, from Step 7 is integrated to find the deviation of the aircraft from it's ideal linear track in the across track direction (perpendicular to the flight direction, positive towards the right wing of the aircraft (the side containing the InSAR antenna)). The result is stored in crosstrack.
10. The vertical displacement of the aircraft, measured relative to it's initial altitude, is determined by integrating vel_vert to give inu_z. If GPS is

available, this is compared with gps_elev (also relative to gps_elev(1)) and a straight line is fitted to the difference. This linear bias is subtracted from inu_z to give inu_z_cor, the unbiased vertical displacement from the start of the line.

11. The angle, drift, is defined as the sample by sample difference between mocomtrack and the filtered maid_f_heding. This is supposed to be, perhaps, the skew angle of the aircraft from its nominal track (with positive towards the LEFT wing or counterclockwise from the track direction). It is expressed in radians.

12. The MAID pitch angle is filtered and converted to radians.

13. The MAID roll angle is filtered and converted to radians. N.B. A lot of the remaining calculations rely on the measurement of specific lever arms and angles on board the aircraft. There is no verification that these are still valid. Several comments in the code do mention specific memos, but they may not be a complete set of the most up to date numbers.

    N.B. SUPER IMPORTANT - The coordinate system set up for all of the internal aircraft displacements and coordinate transforms follows the convention that X is towards the right wing, Y towards the aircraft nose, and Z is UP. The internal Litton coordinate system has Z being down. Since incidence angle is defined between a vector from the radar towards the earth and a vector from the radar towards the target, it also has a positive DOWN coordinate system. It is VERY VERY important to bear this in mind when doing the coordinate transforms and later when doing the mocomp calculations.

    Setting up some constants: (all distances in metres)

    - d1 = 0.286 - Aircraft pitch axis aft of INS reference location,
    - d2a = 0.77 - Aircraft pitch axis below INS reference location,
    - d1b = 0.086 - Aircraft azimuth axis aft of pitch axis,
    - d2b = 0.197 - Aircraft elevation/depression axis below pitch axis,
    - d3 = 0.146 - Aircraft elevation/depression axis towards left wing from azimuth axis,
    - d4 = 0.092 - Main antenna phase centre below elevation/depression axis (ant coords),
    - d5 = 0.276 - Main antenna phase centre right of depression axis (ant coords),
    - d6 = 0.065 - Fixed distance between centre of InSAR antenna and its mount, regardless of how InSAR antenna is steered,

- d71 = 0.22 - Distance forward from centre of InSAR ant to fore phase centre,
- d72 = 0.22 - Distance aft from centre of InSAR ant to aft phase centre,
- Xinsarmount = 1.814 - InSAR antenna mount towards right wing from INS reference,
- Yinsarmount = -0.3732 - InSAR antenna mount forward of INS reference (since this value is negative, the InSAR mount is AFT of INS),
- Zinsarmount = 0.9547 - InSAR antenna mount above INS reference.

Basically, everything is referenced to a nominal linear track defined by the GPS (if available) and the deviations of the reference location of the INU from this track have already been computed in the across track direction in Step 9. Now, everything else is referenced to this location and then transformed to absolute GPS coordinates. The general description of how the INU measures things and of how the reference track concept works (without GPS) is described in the October 16, 1991 document, "The Software-Based Motion Compensation System for the CCRS Airborne SARs: Technical Description V2.0" by P. Vachon and J. Wolfe.

14. The azimuth angle of both the main and InSAR antennas is filtered and converted to radians. Several other angles are also converted to radians.

    Angle Summary:

    - azmain - filtered measured main antenna azimuth (clockwise from nose),
    - azinsar - filtered measured InSAR antenna azimuth,
    - empos - mean measured main antenna depression angle (positive is right wing down, zero is right wing horizontal),
    - eipos - nominal fixed InSAR antenna depression angle,
    - mppos - mean measured main antenna pitch angle (positive is nose up),
    - ippos - nominal fixed InSAR antenna pitch angle (although this is called insar_azi_angle in parameter file, it is really pitch),
    - drift - filtered azimuth angle of INS frame from linear track (positive is counterclockwise from zero when nose is pointing along track),
    - pitch - filtered pitch of INS frame (positive is aircraft nose up),
    - roll - filtered roll of INS frame (positive right wing down from horizontal).

    N.B. Although the main antenna depression angle and pitch angle are available as a function of time, they are replaced everywhere by their mean values. This seems to be a potential problem in turbulent conditions.

The coordinates of the phase centre of the main antenna in the aircraft frame of reference (x - right wing, y - nose, z - up) with the INS reference position as the origin is calculated as (xp,yp,zp). The details of how to perform this calculation (and a similar one for the InSAR antenna phase centre which occurs in Step 16) are described in the August 6, 1991 memo, "Model for InSAR and main antenna phase centres" by L. Gray. The antenna description in this memo supersedes the one in the P. Vachon technical description, even though the latter has a later date. The equations in the memo (and in all sections of this code, except where noted for the GPS to INS transform) have been checked. It is important to note that the definition of angles for the InSAR antenna is confusing. The assumption is that the phase centres are located at (d6,-d72,0) and (d6,d71,0) to start with in the unrotated frame which has an initial azimuth of 90 degrees. We then rotate down by depression angle, clockwise by azimuth angle (bearing in mind it started out at 90 degrees), and finally down by pitch angle. Pitch angle is down because PITCH IS STEERED DOWN TO COMPENSATE FOR AIRCRAFT NOSE UP, SO POSITIVE PITCH STEERING ANGLE IS DOWN.

15. The main antenna coordinates are now converted to displacements from the INS reference position in the frame of the ideal track which is defined by the alongtrack displacement (alongtrack), the crosstrack displacement (ctm) and the displacement in the absolute up (zm2) direction.

16. The position of the phase centre of the complete InSAR antenna in the frame of reference of the InSAR mount is calculated as (xi,yi,zi). The positions of the fore and aft phase centres are computed as (xi1,yi1,zi1) and (xi2,yi2,zi2), respectively. These are in the antenna mount frame and they are converted to the INU frame simply by adding the constants (Xinsarmount,Yinsarmount, Zinsarmount). The tij rotation matrix calculated in step 15 is then applied to move things to the ideal track reference frame. This results in the fore and aft coordinates being (cti1,aalongtracki1,zi21) and (cti2,alongtracki2,zi22), respectively.

    N.B. Steps 17 and 18 are only executed if GPS data is available.

17. The coordinates of the GPS antenna relative to the INU reference position in the aircraft body (INU) frame are defined as (xGPS,yGPS,zGPS) = (0,7.2878,2.1812). The tij rotation matrix is then used to convert this position to the ideal track frame, as (xGPSm,yGPSm,zGPSm).

    N.B. This position comes from Karim Mattar's notes, according to the comment in the code, and has not been independently verified. The rest of these two steps relies on a May 1984 document, "Technical Description of

Litton 90 System", which defines the Litton frame, and also has not been independently verified.

lat and long are defined as gps_lat and gps_long converted to radians. mocomtr is defined as negative mocomtrack, converted to radians. Thus, it is the best linear fit to the GPS track, but defined positive counterclockwise from North.

18. A Cij rotation matrix is derived to transform from the INU frame to a fixed earth ECEF frame defined by:

    - X axis intersects 90 east meridian at equator,
    - Y axis points towards north pole,
    - Z axis intersects Greenwich meridian at equator.

    N.B. VERY IMPORTANT!! Using the two frames of reference from the notes in the code, the rotation matrix may have an error and should be $C31 = -cos(mocom)sin(long) - sin(lat)cos(long)sin(mocom)$. Also, this is a pure rotation without translation, which means it can be used to translate differences in position, but not absolute positions. This is O.K.

    The difference in the ideal track frame between the main antenna phase centre and the GPS antenna is computed as (xdif,ydif,zdif) and rotated to a difference along the three axes of the ECEF frame using the Cij matrix. This gives (Xdif,Ydif,Zdif), the position of the main antenna phase centre relative to the GPS antenna in the ECEF frame. According to another unverified document, the standard output of the Ashtech GPS receivers in (x,y,z) format is cast in a different form with:

    - Y axis intersects 90 east meridian at equator,
    - Z axis points towards north pole,
    - X axis intersects Greenwich meridian at equator.

    This axis swap is made and then the absolute positions from the GPS receiver, (gps_x,gps_y,gps_z), are added to the differences in the same frame, yielding the absolute position of the main antenna phase centre in the GPS ECEF frame. This is defined by (Xant,Yant,Zant).

19. Now the final calculations for the deviations of the antenna phase centres from the ideal track (in the ideal track frame) are made. Specifically:

    - delh_main - Deviation of main antenna in across track direction,
    - delh_ati1 - Deviation of Fore InSAR antenna in across track direction,
    - delh_ati2 - Deviation of Aft InSAR antenna in across track direction,
    - delv_main - Deviation of main antenna in up (Z) direction,

- delv_ati1 - Deviation of Fore InSAR antenna in up (Z) direction,
- delv_ati2 - Deviation of Aft InSAR antenna in up (Z) direction,
- dely_ati - Distance from Aft to Fore InSAR antenna in along track direction.

20. Now the calculated variables are saved in the output file defined by the parameter moc_file in the parameter file.

    If no GPS is available, the filtered MAID groundspeed is copied to gps_vg, since the veloc program assumes a gps_vg exists to give the aircraft speed when computing the target radial velocity. Then, an appropriate set of parameters are saved, depending on whether or not GPS was available to calculate Xant, Yant, Zant and gps_elev.

6.8 landphscal

CCRS Airborne Along Track Interferometric SAR Processing Software Revision 1.0 - As received from CCRS by DREO.

PROGRAM DESCRIPTION - LANDPHSCAL.M
John W. M. Campbell, Aug 12, 1999

This program takes data from the image imagefile, which should be an azimuth sub-scene of the full image described by the parameter file paramfile. This sub-scene should contain only land, which is assumed to have a velocity (and thus an interferogram phase) of zero. There should also not be any significant moving targets in this land sub-scene. A phase profile is generated which can be used to (on average) phase rotate the data to be centred about zero phase. The profile is one dimensional, with a single value for each range pixel in the image, and is saved in the file phscor_file defined in paramfile, as two MATLAB arrays: ang, which contains phase rotation angles, and theta, which contains the corresponding incidence angles.

The calling sequence in MATLAB is:

landphscal (paramfile, imagefile, cutoff)

with paramfile - the processing parameter file created in startATI for the data contained in imagefile
imagefile - an azimuth sub-scene of the data described by paramfile
cutoff - A magnitude cutoff, below which the phase of a pixel is assumed to be unreliable. This value is not used in the program, but is written to paramfile for later reference.

The first thing the program does is to load the following parameters from the parameter file, paramfile.

- range_gate_delay (rgd) - delay before ADC sampling begins on each pulse,
- saw_delay (sawdel) - delay associated with SAW range compression,
- elev_to_ant_Ca (elevCa) - mean elevation of main antenna phase centre,
- phscor_file (phsname) - name of output file to store phase profile,
- near_edge_pixel (fpix) - first range pixel in processed swath,
- far_edge_pixel (lpix) - last range pixel in processed swath.

The program then loads the data from the MATLAB file imagefile. It used to load the PRF file as well and use delh_ati1, which gave the height to the fore InSAR antenna as a function of azimuth position, but it was decided that the slight change in incidence angle due to varying height was not worth worrying about.

N.B. This could be brought back in a future upgrade.

Next, the base range and then the ratio of the average height divided by the range to each pixel is calculated using the speed of light and the timing information, as well as the fact that the range sampling occurs at 75MHz. Any "airball" pixels, where the range is less than the height (*i.e.* the ratio is greater than one), are identified and eliminated.

With the assumption of a flat earth, the incidence angle to each remaining pixel is computed as the arccosine of the ratio.

Next, the phase angle of every pixel in the image is computed. A histogram with fifty bins is computed from the phase angles, and the data is phase rotated to be centred about the maximum of this histogram.

N.B. This stage is not really necessary. It is just a holdover from when regular analysis of the phase was carried out on every scene.

Next, the mean of the real and imaginary components are calculated using the MATLAB mean command which calculates the mean for each column (*i.e.* each range value). The phase angle defined by the mean real and imaginary values is then computed (including the additional peak histogram phase through which the data had already been rotated) to give a final vector of phase angles, ang, which will phase rotate each range column to have a mean phase of zero.

Finally, airball pixels in ang are eliminated, ang is smoothed using the .mex file smooft and both ang and theta are written to phsname. Cutoff is also written to paramfile.

6.9 gpsmat

CCRS Airborne Along Track Interferometric SAR Processing Software Revision 1.0 - As received from CCRS by DREO.

PROGRAM DESCRIPTION - GPSMAT.C
John W. M. Campbell, April 8, 1999

This program takes the ASCII file produced by differential processing of the aircraft and ground GPS and translates it into a MATLAB file for use in further processing.

N.B. The Geogenius software may have slightly different ASCII formats, so some changes may be required to read GPS data. GPSMAT2 from the polarimetry software suite may be a good guide to what changes are necessary.

N.B. The savemat and readmat functions were designed for MATLAB 4.2. Significant changes have occurred to the MATLAB file structure since then, and it would thus be advisable for someone to rewrite these subroutines to reflect MATLAB 5.2 or 5.3. In the meantime, MATLAB should still handle these files if the version flag is specified in all the MATLAB I/O. This will necessitate changing all of the .m files in this software of course. The best solution would be to replace these old I/O routines with the .m and .mex routines now supplied with MATLAB.

N.B. As with much of the software in this suite, many of the subroutines are given either as include files or as extra code at the end of the main program. So that these subroutines only have to be fixed once, they should probably be put in separate files, and a proper make file generated to compile each program.

N.B. The files dates.c, dates.h and strlow.c, which have to be included to compile this program were NOT supplied to DREO, but have been included in the software sent to Atlantis Scientific.

The bulk of the program is reading an ASCII file and saving variables in a MATLAB file. There is nothing complex here, except in the time code.

N.B. The time code read from the GPS ASCII file has only two year digits. 1900 is added to this number, and the time is converted to a NIMLIB time format (seconds since Jan 01, 1990). This is NOT Y2K compliant.

6.10 insarmaid

CCRS Airborne Along Track Interferometric SAR Processing Software Revision 1.0 - As received from CCRS by DREO.

PROGRAM DESCRIPTION - INSARMAID.C
John W. M. Campbell, April 13, 1999

The documentation supplied with this program (shown below) is almost negligible.

This program is run to convert the MAID data in the raw file format as it was copied from exabyte tape by dd into a MATLAB format file.

The calling format is:

INSARMAID infile.dat outfile.mat

where infile.dat is the name of the binary MAID file created with dd, outfile.mat is the name of the MATLAB file to be created.

N.B. This program uses the exact format of the MAID data as it was at the last time the program was used. Any changes whatsoever will really mess it up since it does a lot of buffer counting and looks for specific formats in specific places.

N.B. Most data is left in its native units (often imperial units), but maid_f_gndspd is converted to m/s.

If the MAID format hasn't changed, this program should work provided two problems are addressed:

1. Changes to 64-bit compiler will require explicit variable initialization (memset, *e.t.c.*) and may have affected things like size_t. This will all need to be checked carefully during compilation.
2. The MATLAB file is created using the infamous imglib. That means that it is different again from the readmat/savemat subroutines or the readimatd/saveimat subroutines used in some of the other programs. Thus, we are dealing with three different MATLAB I/O routines. There is a strong need to standardize on one MATLAB I/O system and make it 5.2 compatible. We should probably implement the .m and .mex libraries supplied with MATLAB. In the interim, imglib is probably working since it is required for the polarimetry software. So, if there are problems, the imglib supplied with the polarimetry software could be compiled with this program.

6.11 sigdmp

CCRS Airborne Along Track Interferometric SAR Processing Software Revision 1.0 - As received from CCRS by DREO.

PROGRAM DESCRIPTION - SIGDMP.C
John W. M. Campbell, April 13, 1999

The documentation supplied with this program (shown below) is unusually good for the CCRS software in the AT-InSAR set.

N.B. It appears that this software has already been updated to memset the array variables and initialize other variables to zero.

N.B. The portion of the program which offers a selection of default device names can easily be modified for the local system (but it isn't essential).

This program needs to be run four times, once for each of the four channels of AT-InSAR signal data which has been stripped to exabyte tape. Usually, only a single flight line will be on each exabyte. Since the tapes were created by stripping from HSR, they won't have a tape identifier to use for naming. A naming convention like: PCU1A, PCU2A, PCU1B, PCU2B is suggested for the four tapes and the corresponding files created by this program.

It is best to extract data by record number, since the same number of records must be extracted for each of the four channels.

N.B. ******* Due to the UNIDSK heritage of some of the later programs, there will be problems if the number of complex samples extracted is not a multiple of 512. There will also be problems (due to bad programming) if the number of records extracted is not a multiple of 512. ********

N.B. Both raw and ancillary data are required.

Although sigdmp2 and sigdmp4 from the polarimetry set appear to be newer than this code, the presence of the memset statements suggests that this code was updated for a 64-bit IRIX. Since the coarse attenuation is not required for the along track InSAR and since there is uncertainty whether sigdmp4 produces the same format of output files, retaining sigdmp is recommended.

6.12 startati

CCRS Airborne Along Track Interferometric SAR Processing Software Revision 1.0 - As received from CCRS by DREO.

PROGRAM DESCRIPTION - STARTATI.M

John W. M. Campbell, April 14, 1999

The design of this program appears to be rather mindless. It forces the user to specify an input and output file for the first step (which makes its warning that the input file will be overwritten rather irrelevant) and then it asks for a new input and output file for the second step. Obviously, the input file in the second step and the output file from the first step must be the same.

Anyway, the first step simply chops off the beginning and end of all the MAID parameters in the MATLAB file created by insarmaid, using a start and end index to the maid_f_roll parameter given by the user. It then saves the truncated data to an output MATLAB file.

The second step reads the data back from the truncated file and splines every variable to a common 64 Hz time code using the .mex file cspline. This .mex file has not been provided with the InSAR code, so using the version from the polarimetry code, which appears to be the same, is suggested. Hopefully, Atlantis Scientific has already compiled this as a .mex file with a 64-bit SGI compiler. If not, this will have to be done.

The third step is the interactive generation of a parameter file. Some of this data (flight data, line, pass, etc.) is for information purposes only, but some of it is relevant to the files already created (*e.g.* maid_file just created in Step 2, raw_data_?? created by sigdmp) or files which will be created later in the processing chain. Other values are parameters which maybe available from the log sheets, from the ancillary file (use dispanc to view it) or which may have to be calculated externally. The suggested values in the program are reasonable first guesses, except that some, such as gps_moc_delay, seem to vary a lot from mission to mission.

Note that the used_gps should always be "y" unless GPS collection failed for some reason. Similarly, used_ivspeed and main_ant_disabled should both be "1" unless a special circumstance (such as an antenna motor failure) occurred. If such a special circumstance does occur, then the software will probably have to be modified anyway, but at least some of the right hooks are in place.

N.B. The program suggests that setting used_ivspeed = 1 will cause ivspeed NOT to be used. In fact, just the opposite is true. If used_ivspeed = 1, ivspeed WILL be used, which it has to, because the vertical acceleration parameters are not carried through the processing chain in this version.

N.B. The program asks for an azimuth angle for the InSAR antenna, called insar_azi_angle, with a suggested value of 2.0. This should actually be the pitch steering angle for the InSAR antenna.

N.B. The max_ram should be set to half of the RAM available on the machine and the max_cache to half of that.

N.B. The range and azimuth pix_offset parameters must be checked for each mission to ensure coherence between the channels and that the correct interleaving of odd and even pulses is being used.

N.B. All of the MATLAB programs in the set use .mex files argmscan and argmwrte to read and write from the parameter file.

The UTM parameters will be used to generate the extent of the North Up presentation of the data. They can be small to select a target sub-scene, but initially it is best to let them be larger than necessary so all of the scene is visible.

The smooth_factor and overlap_factor determine how many SLC samples are averaged to get each output sample (in the range direction). This will be described in detail later.

6.13 atinsarnew

CCRS Airborne Along Track Interferometric SAR Processing Software Revision 1.0 - As received from CCRS by DREO.

PROGRAM DESCRIPTION - ATINSARNEW.C
John W. M. Campbell, June 10, 1999

This is the main SAR processing program which generates a single file containing the real and imaginary parts of the interferogram (antenna1 * complex conjugate of antenna2 on a pixel by pixel basis). If the compiler flag OUTPUT_FORE_AFT is set, the program also produces two complex output image files, one for the fore antenna and one for the aft antenna.

Unfortunately, it is an overly complex program, with some sections of code that are no longer relevant, and a lot of confusing variable names.

It begins by writing its own name to the parameter file (since there were several different versions of this program for different specific purposes) and by reading the names of the various input and output files. It attempts to open the output log file, the output data file, and if OUTPUT_FORE_AFT, the fore and aft antenna image files which have hard-coded names "fore.mat" and "aft.mat".

The program then reads a number of parameters required to set up the parameters for the SAR processing itself. Specifically, it calculates:

Base Range - rbase = (rgd-delay1) * hc - delsr, where rgd is the range gate delay and delay1 the SAW delay caused by the finite time required for real time range processing using a SAW device. These are given in microseconds, where hc is half the speed of light in meters per microsecond, delsr is the slant range pixel spacing defined by the sampling rate of 37.5 MHz. Thus, rbase is the range from the bistatic phase centre (half way between the main antenna and either half InSAR antenna) to the zeroth targetpixel.

Base Height - h = rbase + nadir*delsr - delvinsa - 0.5*(delvinsCa-delvinsa), where nadir is the index of the first pixel which shows a return from the ground (a user input parameter which often has to be calculated by initial processing of a small sub-scene). delvinsa is the mean deviation of the main antenna from the ideal track in the vertical direction and similarly, delvinsCA is the mean deviation of the fore InSAR antenna. The concept is to go from the height relative to the bistatic phase centre to a height relative to a known quantity such as the reference centre for the INS track. The height to the fore InSAR antenna phase centre, elev_to_ant_Ca, is then h+delvinsCa and the height to the main antenna is h+delvinsa.

N.B. If the sub-scene used to measure the nadir pixel is over a portion of the track for which the vertical antenna deviations stray significantly from their means (*i.e.* if the aircraft is flying too high or low at that point), then the calculations above will be offset. A better solution would be to compute h using a specific subset of delv_main and delv_ati1, which was appropriate for the scene used to measure nadir. This would help account for aircraft altitude differences, but would still ignore terrain height changes. Both aircraft altitude and terrain height changes are probably secondary effects, but future modifications should attempt to address them.

Next, the program opens the four signal data files using the raw data file names from the parameter file. Ca has variables with no numerical suffix, Cb with the suffix 2, Xa with the suffix 3 and Xb with the suffix 4. Error checking ensures that all four files have exactly the same number of lines and pixels. The first and last lines and pixels of the sub-scene to be processed are then read from the parameter file and error checking verifies they are within the bounds of the file (although NOT that p1 < p2 and l1 < l2).

Several important parameters are now calculated:

- mpix = p2-p1+1 - The number of range pixels to process at each azimuth pulse,

- p1a - The first pixel to process at each azimuth pulse,
- pp1a = 2*p1-1 - The first byte to process in each complex range line,
- dsprCbi - Azimuth pulse offset of Ca channel relative to Cb channel,
- dsprXai - Azimuth pulse offset of Ca channel relative to Xa channel,
- dsprXbi - Azimuth pulse offset of Ca channel relative to Xb channel,
- All of these three values give the integer number of azimuth pulses which the first record of Ca data is ahead of the corresponding data.

The start and end lines for all four of the signal data channels are adjusted to accommodate these azimuth offsets.

A similar calculation occurs with the user input parameters for the range offsets, which were input during startATI, except that we have:

- dspCbi - Integer range pixel offset of Ca downrange of Cb,
- dspXai - Integer range pixel offset of Ca downrange of Xa,
- dspXbi - Integer range pixel offset of Ca downrange of Xb,

and instead of shifting both the first line and last line to be processed, only the first range pixel (represented by p1a and the more specific pp1Ca, pp1Cb, pp1Xa and pp1Xb, the starting bytes for range in each channel) and the total number of range pixels to process are adjusted. In fact, mpix = mpix - abs(min(0,dspCbi,dspXai,dspXbi), which means the final pixel to be processed should never be larger than it was before the offsets were computed (at least for the Ca channel) but there may be fewer pixels processed than the user requested or the final pixel may be too large in other channels.

N.B. This entire procedure could potentially lead to lines or pixels which are after the end of the file (or of their individual azimuth pulse, in the case of pixels). A series of error checks for these conditions should be inserted at this point in the code.

The pixel spacing is now set to half of its computed value to account for the interleaving of the fore and aft channels which will occur in this program.

The next set of constants relate to the reference function generation for azimuth convolution (*i.e.* SAR doppler processing). The beam is assumed to be broadside, centred at zero-doppler, which is also broadside, and covering only a small angular beamwidth so we can use small angle approximations and assume no range walk. The earth is assumed to be flat and stationary. The constants are:

- phi - The beam width to be processed (in degrees),
- r - The range to the first pixel to be processed,
- temp2 - The arc delineated by the half beam at the farthest range to be processed divided by the pixel spacing,
- maxrefsz - The size of a symmetric convolution reference function at the far range,
- nfft - The smallest power of two larger than 2*mlin+maxrefsz-1, where mlin = l2-l1+1. This is thus the smallest power of two large enough to accommodate the interleaved data plus the zero padding required for convolution with the reference function.

The program then asks the user if the azimuth processing can use nfft-maxrefsz+1 pixels instead of mlin pixels. Actually, this is WRONG, since the former refers to true interleaved pixels, whereas mlin relates to azimuth pixels from each of the un-interleaved data files. Anyway, if the user agrees, the program changes mlin to be (nfft-maxrefsz+1)/2, which is correct, and the end lines for each of the four channels are adjusted accordingly.

Next, the program determines how many range pixels can be loaded into max_ram and processed at one time, allowing for fore and aft interleaved data to be read into double precision complex arrays. This is npix, and it then defines pexloops, the number of processing loops required with npix pixels, and rempex, the number of pixels in the final (smaller) loop (if it is required). remflag is set to 1 if a final small loop is required.

Similarly, the quantity of raw byte data which can be read at one time into max_cache (or a smaller disk cache of size dcas if the subroutine assin finds that there is insufficient memory available to support max_cache) for corner turning is defined by:

- nlin - number of range lines (constant azimuth pulse number) to read,
- linloops - number of loops with nlin lines required to read in all data,
- linflag - set to 1 if a separate smaller loop required,
- remlin - number of range lines in smaller final loop.

N.B. There is an error here. Although the program checks for sufficient memory and, if unavailable, claims it will use dcas 16 bit samples instead of DCASHSIZE samples, it then does the calculations for nlin, *e.t.c.* based on

DCASHSIZE, regardless of how much memory was available. It should use dcas instead.

Now, some of the constants required for Gaussian smoothing of the output interferogram in order to reduce speckle (effectively multi-looking by coherent averaging) are computed. They are

- mlin2 - number of interleaved samples to be used in processing,

  N.B. Maxrefsz is subtracted from this, because in the final output image, the start and end of each azimuth line will be truncated to produce a rectangular rather than trapezoidal image.

- ttemp - 3dB width (in pixels) of normalized Gaussian used in smoothing

- hddtout - nominal pulse record number on HDDT which corresponds to first pixel in the interferogram (accounting for processing and averaging loss). This pulse is really the centre (hopefully zero-doppler) pulse of the synthetic aperture of the centre pixel of those pixels averaged to produce the first output pixel (at far range of the processed scene). At near range of the scene, this number would be smaller. The appropriate hddt record for any output pixel can then be determined by adding the number of lines in the output image between it and the first pixel at far range multiplied by subsample (smooth_factor from the parameter file).

- clin - number of lines in output image after smoothing,

- sa,gamma,etc. - a bunch of confusing constants that distract from the true nature of the weighting function. It is simply a normalized Gaussian with 3dB width ttemp and total length 2*ttemp+1. Since the centre of each successive Gaussian is shifted by subsample pixels (where ttemp = subsample * resub), the factor resub defines the number of output pixels on each side of a given pixel which share any un-smoothed pixels with it at the 3dB width or higher.

At this point, the weightd vector for Gaussian smoothing and the ougsi and ougsq vectors, which will contain each output image line (at constant range) with real and imaginary components respectively, are defined. The weightd vector is also populated with the normalized Gaussian weightings or, if subsample < 2, with ones.

N.B. This is because a single look complex image is obtained using subsample = 1 and resub = 0.

Now the output interferogram image file is created. If the OUTPUT_FORE_AFT flag is set, two extra files for the fore and aft SAR images are also created.

N.B. All of these files are created with the hideous openimat, which requires a separate pair of global variable pointers for each MATLAB file we are going to work with and uses an if loop in the openimat subroutine itself to determine which pointer is relevant.

Next, the channel balancing gains are computed using the means and means of squares calculated for the real and imaginary parts of each channel in the meandati program. The assumptions and algorithm work as follows:

For a given PCU, channel and component (say C, A, real) let:

1. rCa - be the measured and recorded signal data over the image, rCa = GrCa * rCa_t + OrCa,
2. rCa_t - be the true signal data, which would have been recorded in system with no amplifier gain or bias,
3. GrCa - be the amplifier gain for the real component of channel A, PCU C,
4. OrCa - be the amplifier offset or bias.

Now, let:

1. mnrealCa - be the measured mean of the signal data over a given subscene,
2. mnsqrealCa - be the measured mean of the squared signal data over the subscene,
3. mnrealCa_t - be the mean of the true signal data over the subscene,
4. mnsqrealCa_t - be the mean of the squared true signal data over the subscene.

Since the real and imaginary components of the signal data will vary rapidly from pixel to pixel as the phase changes, the mean of either one over the subscene should be zero. We assume that GrCa and OrCa do not vary across the subscene, which is generally true if the system remains linear. Then,

mnrealCa = GrCa * mnrealCa_t + OrCa = GrCa * 0.0 + OrCa = OrCa

Now consider the square of the measured data

$rCa^2 = (GrCa * rCa\_t + OrCa)^2 = GrCa^2 * rCa\_t^2 + 2 * GrCa * rCa\_t * OrCa + OrCa^2$.

We take the mean of both sides of this equation over the subscene

$mnsqrealCa = GrCa^2 * mnsqrealCa\_t + 2 * GrCa * mnrealCa\_t * OrCa + OrCa^2 = GrCa^2 * mnsqrealCa\_t + 0.0 + mnrealCa^2$.

Thus, $GrCa = sqrt[(mnsqrealCa - mnrealCa^2) / (mnsqrealCa\_t)]$

The mean of the squares of the real or imaginary components from either of the odd or even pulses from either of the half-antennas is simply a measure of the total radar signal returned from all of the scatterers in the averaged subscene. Thus, it should be the same for all eight components. Since we only require a relative scaling of the eight data streams in order that they can be used together to form the final image, we will assign this factor to be unity. Then,

$rCa\_t = (rCa - OrCa) / GrCa = (rCa - mnrealCa) / sqrt(mnsqrealCa - mnrealCa^2)$.

The same applies to iCa, rCb, iCb, *e.t.c.*

In the next code segment, entitled Part 2, the program opens the prf_file created by ATIPRF and attempts to read the vectors containing the deviations of the various antenna phase centre from the ideal track in the horizontal (across track) and vertical direction. These vectors are:

- delv_main - Vertical deviation of main antenna phase centre from ideal track,
- delh_main - Horizontal deviation of main antenna phase centre from ideal track,
- delv_ati1 - Vertical deviation of Fore InSAR antenna phase centre from track.
- delh_ati1 - Horizontal deviation of Fore InSAR antenna phase centre from track,
- delv_ati2 - Vertical deviation of Aft InSAR antenna phase centre from track,
- delh_ati2 - Horizontal deviation of Aft InSAR antenna phase centre from track.

Then, the program calculates the offset into the mocomp data (which was splined up to the interleaved time stamp in atitimetestnew) and calls this xstart. It then does some error checks by comparing the length of the vectors (the two

InSAR antennas are assumed to have vectors of the same length) to mlin, the number of lines to be processed. Actually, this makes no sense, since mlin is number of lines per channel, NOT number of interleaved lines. Also, the mocomp vectors are for the entire segment which was processed through ATIPRF, and we may just be processing a subscene of that. Thus, this error check should be removed.

Next, the 2-D matrices for the output SAR images from the fore and aft InSAR antennas (fltout_a and fltout_b respectively) are allocated, using the subroutine ftmatrix, which simply uses calloc to create an array of row pointers and then again to create an array for each row.

N.B. There are both strengths and weaknesses in this approach, so it is not clear whether this approach is a good one or not.

If sufficient memory was available, it goes on to allocate the far smaller arrays for the SAR processing (which uses the old SLIP algorithm). This uses the subroutines fvector and dvector, which also use calloc. Next, the arrays to be used in doing the range and azimuth resampling are allocated. Since the InSAR mode of the radar can't have more than 2048 pixels, 2056 represents the maximum possible value of npix+8. It is assumed that there will never be range resampling of more than 8 pixels, which is reasonable.

The arrays delresCb, delresXa, and delresXb are set up as the range pixel indices of the Cb, Xa, and Xb channels, assuming that Ca runs from 1 to 2048, and using the relative range offsets which were user input back in startATI. These should have been found by some signal data correlation tests. Hopefully, they are stable from line to line, so one set of correlation tests per flight will be adequate. These user variables, dspCbf, dspXaf and dspXbf were also used in calculating the pixel integer offsets earlier in the program.

A few other miscellaneous variables are initialized to zero and then the main processing loop begins.

The outer loop (variable ii) is for the disk reads of the input data. Since the corner turn and SAR processing takes place here, the data is tiled and processed by tiles. This outer loop goes through the pixels, which are broken into pexloops of npix pixels and a possible remainder loop with rempex pixels.

First, npix is copied into the variable ppix. Then, if this is the remainder loop, npix is set to rempex. Copying npix to ppix does not need to be inside the for loop. This only needs to be done once.

Each channel is now handled separately, in the order Ca, Cb, Xa, Xb. The start byte of the current loop is given by pp1, which takes into account the integer range offsets calculated previously (pp1Ca,pp1Cb,etc.). For the channels after Ca, where there may have to be splining in range, several other steps have to take place. These will be indicated by a ** symbol. [8] [9]

N.B. Instead of npii+p1-1 > p2, we should really compare to nblks * 256, the total number of pixels in a range line. [10]

The jj loop for each channel goes through groups of llin lines with a final remainder loop (if necessary) of remlin lines. The byte offset into the file is computed and the disk cache byte array, cash, is filled starting at that point in the file. This means that all range pixels on every line in the current jj loop are read into cash.

The i loop for each channel loops through the lines read inside the current jj loop so they can be corner turned, balanced, and, if necessary, splined in range.

jlin = 4*i + 4*(jj-1)*llin - This sets jlin to be four times the line number (measured relative to the first line to be processed in the first jj loop). jlin and jlin + 1 will be the I and Q line indices for the C channel data for each antenna, whereas jlin + 2 and jlin + 3 will be for the X channel data. This interleaves the data correctly with I and Q being separated by line rather than by pixel.

The j loop moves through all pixels in the current ii loop on line i and copies them into the array fltout_a[pixel number][line number] for real values or fltout_a[pixel number][line number + 1] for imaginary values. This accomplishes the corner turn. The values from cash are also channel balanced at this point, using mnrealCa, mnrealCb, gaincar, gaincbr, *e.t.c.*, and the algorithm described earlier. [11]

N.B. The output flttmp will usually be larger than the portion we want because of the extra bytes read on either end for splining. The solution is that the j2loop

[8] 8 bytes are subtracted from pp1 and 8 pixels added to npix (npii = npix + 8). This gives a buffer of 8 extra bytes on each range line, which may be required for splining. spee is initially set to 4, the position of the original first pixel in the new expanded range array.

[9] If the first pixel is now less than 1 or the last pixel is greater than p2, the larger array size for splining is reduced and npii, spee, and pp1 are adjusted appropriately.

[10] The array dresCb (or dresXa or dresXb) is set up with indices from 0 to npii - 1 given by 1 + dspCbf to npii + dspCbf (or dspXaf or dspXbf). This is done through a needlessly complex reference to the previous array delresCb (or delresXa or delresXb). Presumably, there was a valid historical reason for this extra array reference.

[11] Since channels Cb, Xa, and Xb all have to be potentially splined in range prior to being placed in the fltout_a and fltout_b arrays, they are initially copied to the single line vectors fltbr and fltbi for the real and imaginary parts, respectively. The spline2 subroutine is then used to spline each one into the variable flttmp, using the offset index variable dresCb (or dresXa or dresXb). Finally, flttmp is copied into the appropriate line of fltout_a (or fltout_b).

which copies flttmp into fltout_a starts reading npix (not npii) pixels at index spee into the flttmp array. This is fine, BUT THERE IS AN INDEX PROBLEM. The integer portion of dspCbf (or dspXaf, etc.) has already been taken care of by the shifting around of the portion of the input data to be processed, which was done in creating pp1Ca, pp1Cb, etc. Thus, to spline by the full value of dspCbf *e.t.c.* means the data will have been shifted too much. The solution is to create the array dresCb (or dresXa or dresXb) to be from 1 to npii, with an offset of (dspCbf - dspCbi) (or the equivalent for Xa and Xb).

Now all the data for the current ii loop of pixels has been read, balanced, interleaved, and range splined. Next is the phase adjustment for the MOCOMP processing. This is done by looping ilin from 0 to 4*mlin in steps of 2 (for I and Q). For each line, the following simple algorithm is used:

delind = ilin/2+xstart - current index into the mocomp variables.

Using a variant of the ideal track coordinate frame (specifically, origin at nominal position of ideal track through INS, x-axis vertically up, y-axis in across track direction – generally towards the InSAR antenna, if the skew angle is not too large – and z-axis along the ideal track in the direction of aircraft motion), the three sides and one of the angles in the triangles defined by the origin and the phase centres of each of the main antenna and the fore and aft InSAR antennas (postscripts a, 1 and 2, respectively, on the variables in the code) are calculated.

For each pixel in the current line, for which these triangles have been determined, range to the pixel centre is computed. The height of the main antenna is also computed as h+dv, and thus, with a local flat earth approximation, the incidence angle from the main antenna is acos((h+dv)/r).

N.B. The true path length is from the main antenna to the target pixel and back to the InSAR antenna, and the r which is measured by the delay length is really this path length divided by two. Due to the natural uncertainties in computing delay lengths, however, there will always be some uncertainty in r, and due to uncertainties in terrain height, there will always be some uncertainty in the incidence angle, thetaa. It is only important that we get the incidence angle approximately right. Therefore, this method of calculating the phase offsets should be OK most of the time.

Anyway, the reference range, rref, from the target pixel to the ideal track origin position is now easily calculated using the cosine rule for triangles and then another application of the cosine rule using rref and the incidence angle to the ideal track origin, allows the range to the phase centres of the two InSAR antennas to be calculated, as rb1 and rb2, respectively.

We seek to phase shift everything to the phase delay corresponding to the reference range of the ideal track, (*i.e.* a path length of 2*rref), whereas the measured phase corresponds to the actual path length of r + rb1 (or r + rb2 for the aft antenna). Thus, the phase shift for the InSAR antenna is given by the reference path length - the actual path length multiplied by the phase delay per meter of extra path length. Thus,

delrb1 = -2*pi/lambda * (rref - r + rref - rb1),
delrb2 = -2*pi/lambda * (rref - r + rref - rb2).

The program then simply complex multiplies the phase offset with the complex value for the current line and pixel from each antenna.

After all lines up to 4*mlin have been phase corrected for mocomp in both the fltout_a and fltout_b arrays, these arrays are padded with zeros out to 2*nfft to allow sufficient zero padding for the complex FFT convolution as per Numerical Recipes in Fortran, 2nd Edition, Chapter 13.1.

Part 3 of the code deals with the SAR processing.

Actually, this is relatively simple. The j2 loop goes from 0 to npix, the number of pixels in the current piece of data loaded into fltout_a and fltout_b. j3 = j2 + 1 + (ii-1)*ppix is the output pixel in the output image on disk. i = p1a + j2 + (ii-1)*ppix is the input pixel in the original image, which is used to calculate the range to the target pixel, r, just as it was in the mocomp calculation.

The reference function for the convolution is computed in the spatial domain. The variable temp[jj] = x*x*i2*i2*2*pi/lambda/r is simply the first order Maclaurin Series expansion of the phase delay caused by the range to the target at each azimuth pulse minus the range to the target at zero-doppler (*i.e.* the phase is forced to zero at zero-doppler). The full version of this is:

phase = -4*pi/lambda * [(rref*rref + x*x*i2*i2)$\hat{0}$.5 - rref]

Since the matched filter always uses the complex conjugate of the signal for an ideal reference target, the program actually uses positive phase (*i.e.* -4 is changed to +4) and uses r instead of rref. This will cause a small defocus error, but it may not be significant. The rest of the difference between this phase and temp is simply due to the Maclaurin expansion. Note that i2 begins negative and goes positive, which is probably backward, but the phase function is normally symmetric, so it shouldn't matter.

N.B. dn is the parameter delta_n_refgen. This was put in to allow a slight squint in the processor if desired. Normally, squint, range migration, etc. are ignored by this processor. If dn is non-zero, the fact that i2 goes from negative to positive will be important.

The magnitude of the reference function is stored in weights and is simply a sine function extending from 0 to pi. This is identical to a cosine from -pi/2 to pi/2, which is a more typical form.

refgen is then created as a complex function by combining weights and temp then dividing by the normalization factor norm, which ensures the integral of the magnitude is unity.

The function convlc2 then performs the FFT based complex convolution. It uses the algorithm from Numerical Recipes, suitably modified to handle two convolutions at once (so fltout_a and fltout_b can both be convolved with only one FFT of refgen) and also modified to handle complex data.

N.B. The SGI provided FFT routines are MUCH faster than this algorithm and it would be highly beneficial to migrate this program to use them.

Next, the output vector from the aft channel is splined ahead in azimuth by 2*azfr (another parameter determined manually and entered during startATI) using the spline2 routine. Since spline2 doesn't handle complex data, ou2bf is broken into the real and complex parts, fltbrl and fltbil, where each of these is handled separately. 2*azfr is used because it is assumed that azfr was determined by correlation tests on the signal data from individual A and B channels in single PRF form and now double PRF is required.

Now, the interferogram for all the lines at the current range pixel is generated. A subset of the output buffer is used, corresponding to a rectangular output image. For each line within that subset, the real part (of the complex conjugate of the aft antenna) times (the fore antenna) is stored in cdr and the imaginary part in sdr. They are then copied into the output array outfnl in real, imaginary, real–imaginary alternating order. The sums of the magnitudes of each of the fore and aft pixels and the sums of the real and imaginary parts of the complex conjugate are also stored.

The downgaus subroutine is used to do Gaussian averaging on the data, using the algorithm described earlier, and this results in the final ougsi and ougsq averaged real and imaginary output vectors for the current range pixel. These are saved in the output file using saveimat.

If the OUTPUT_FORE_AFT flag is set, the same rectangular extraction is done upon the data in ou1bf and ou2bf, but the data is written back into ou1bf and ou2bf, which are then Gaussian averaged and written out to their respective fore and aft antenna output files.

This concludes the j2 loop for SAR processing the pixels in the current ii loop. The next bracket also concludes the ii loop.

Now, all that remains is to compute the coherence and write it to the log file.

6.14 Motion Compensation

Coherent phase measurement is an essential component of the SAR image formation process. A physical limitation to such measurement is the unavoidable turbulence to which an aircraft is susceptible. Improved coherent phase measurement (for wavelengths significantly smaller than the variance in the displacement of an aircraft from its ideal flight path) can be accomplished by continually measuring the position of the aircraft with accurate motion sensors, and then compensating for any deviations from the ideal flight path.

The Convair 580 possesses a system designed to apply real-time phase correction, but the system is sub-optimal in that it does not work properly. The system hardware setup introduces a further phase error. The system can be turned off, “mocomp off”, and post-flight motion compensation can be done in a software (or other) environment by measurement of the aircraft position at regular, high-frequency intervals.

Motion sensors on board the Convair 580 measure parameters of the aircraft attitude, position and velocity at various frequencies, the maximum being 64 Hz. In addition, GPS measurements provide secondary estimates of the aircraft position and velocity, but the GPS measurements are recorded at much lower frequencies, typically 1 Hz. The two systems of measurement are mechanically quite different, but they physically complement one another.

6.14.1 The INS system and the GPS system

It is beyond the scope of this note to describe the exact mechanism for recording and implementing motion sensing and compensation. As a start, however, we shall outline the differences between the two measurement systems and describe how they may be jointly used.

First of all, there are two different systems (not only are they physically are different, but they measure different sources of information) on board the aircraft.

The maid INS system is an inertial measurement system, which means that accelerations are measured by accelerometers, and a second order initial value problem is solved for the aircraft position. As always, in solving an initial value problem, the projected position and velocity are very much dependent upon initial conditions. Furthermore, because errors are propagated through the system, as well as initial conditions, the system will eventually diverge in accuracy. An estimate of the time evolution of error was suggested as 1 nautical mile per hour.

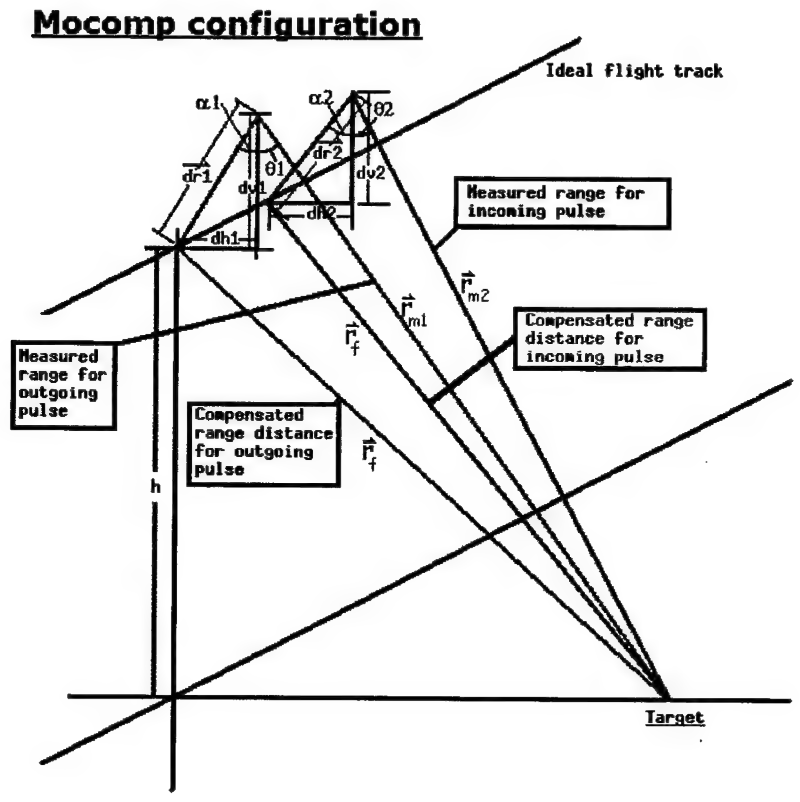
Accelerations that are measured on board the aircraft must be converted from the aircraft reference frame into a ground based reference frame. Once the ground based accelerations are known, the velocity and position evolutions through time are calculated. In doing the conversion from the aircraft body reference frame into the ground reference frame, one of the effects that has to be accounted for is the Coriolis acceleration due to the rotation of the earth.

The GPS system estimates position by solving a geometrical problem involving the observed distance from the receiver to a set of GPS satellites. The rate at which the GPS system can measure the position of the receiver is not rapid enough to track the pulse repetition frequency and is, in fact, not adequate for interpolation of the aircraft position between measurements at spacings on the order of the prf.

The GPS system can, however, measure the position of the aircraft very accurately, though at low frequency. It is, luckily, free of the error that would propagate through the INS system. The idea, then, about combining the two measurements is a good one. Essentially, the low frequency measurements by the GPS system provide updated initial conditions for the INS system, thereby creating a lower bound on the error that could accumulate in the INS system. Furthermore, the quality of the measurement of the INS system over time periods defined by the GPS recording frequency is very good, a win–win situation.

6.14.2 Motion compensation geometry

An ideal flight track can be determined from the measured position of the aircraft over time by performing a least squares fit, or in the case of repeat-pass interferometry, requiring that the second track run parallel to the first. Once determined, the ideal flight track can be used to determine horizontal and vertical displacements from the ideal path. Figure 5 illustrates the geometry of the motion compensation procedure



Mocomp configuration
Ideal flight track
α1
α2
θ1
θ2
dr1
dr2
dv1
dv2
dh1
dh2
Measured range for incoming pulse
$\vec{r}_{m2}$
$\vec{r}_{m1}$
$\vec{r}_f$
Compensated range distance for incoming pulse
Measured range for outgoing pulse
Compensated range distance for outgoing pulse
h
Target

Figure 5: Motion compensation geometry

Motion compensation involves correcting the phase so that it appears as though the phase was measured from the ideal track that the aircraft "should" have followed. The extra distance, either positive or negative, between the aircraft and the target compared to what it ideally should have been is converted into cycles of phase and subtracted from the measured phase.

The maid INS measurements of the aircraft position and the GPS measurements of position are combined and a calculation of the horizontal and vertical displacements from the ideal track is done. The results are tabulated for each pulse in a post-flight software environment. Some of the post-flight, pre-processing software is described in the processor HOW-TO.

What follows here is the geometry of the motion compensation and how it is applied to the raw signal data. If the ideal reference length for the outgoing pulse is r_f, if it is the same for the incoming pulse, if the measured range for the outgoing pulse is r_o and if the measured range for the incoming pulse is r_i, then the extra length traveled by the signal is

$$\Delta r = r_{m1} + r_{m2} - 2r_f. \tag{1}$$

The extra length is converted into phase units by

$$\Delta\theta = \frac{2\pi}{\lambda}. \tag{2}$$

Thus, the measured phase is corrected for by multiplying the complex received signal,

$$z_f = z_z e^{-i*\Delta\theta}. \tag{3}$$

From the diagram, figure 5, the incidence angle is

$$\theta_1 = \arccos\frac{h + dh_1}{r_{m1}}, \tag{4}$$

and the angle α_1 is given by

$$\alpha_1 = \arctan\frac{dh_1}{dv_1}. \tag{5}$$

Once α_1 and θ_1 are determined, the ideal path length is

$$r_f = \sqrt{dr_1^2 + r_{m1}^2 - 2dr_1 r_{m1}\cos(\alpha_1 + \theta_1)}, \tag{6}$$

where dr_1 can be determined from the Pythagorean theorem. With the ideal path length determined, the distance r_{m2} is then, approximately,

$$r_{m2} = \sqrt{dr_2^2 + r_f^2 + 2dr_2 r_f\cos(\alpha_2 + \theta_1)}. \tag{7}$$

The same procedure can be applied to motion compensate the aft antenna. The airborne along track insar processor does the motion compensation calculation as well as the phase correction rotation.

6.15 Smoothing and Overlap

The Airborne INSAR processor (*atinsar1.4*) does some decimation/detection of the SLC data once it has been processed. The parameters controlling the way that the decimation is conducted are the *smooth_factor* and *overlap_factor* in the parameter file.

Smoothing is done with a Gaussian weighting function with standard deviation proportional to a hard-coded value; specifically, $\sigma \propto 1/2.354$. Along with an *overlap_factor* of one, the reason for the specific value for σ becomes clear.

The number of samples that will be included in the decimation operation for each pixel is given by the *smooth_factor* and the *overlap_factor*. The spacing between peaks for the weighting function (as in figure 6) is given by the *smooth_factor*. The default value for the *smooth_factor* is 20 because the pixel spacing in the azimuth direction is approximately 19cm when the radar system is run at a prf/v ratio of 5.14, and the range pixel spacing is about 4m; thus, the overall pixel geometry is approximately square.

6.15.1 The weighting function

The exact functional representation of the Gaussian used for decimation is given by

$$f(x) = \begin{cases} Ae^{\frac{-x^2}{2(L/2.354)^2}} & x \in [-L, L], \\ 0 & \text{elsewhere,} \end{cases} \tag{8}$$

where A is a normalization factor, y is a distance in meters and

$$L = smooth\_factor \cdot overlap\_factor. \tag{9}$$

In terms of dimensionless units on the unit interval, ($y = x/L$), the above may be written

$$f(y) = \begin{cases} Ae^{\frac{-y^2}{2/(2.354)^2}} & y \in [-1, 1], \\ 0 & \text{elsewhere.} \end{cases} \tag{10}$$

The first pixel in the decimated image is calculated as the inner product of the weighting function and the data over the interval $x \in [-L, L]$. Then, the neighboring pixel in the decimated image is calculated as the inner product of data over the interval

$y \in [smooth\_factor - L, smooth\_factor + L]$

and the same Gaussian weighting function as in equation [10], but shifted over by *smooth_factor* units. The center of each set of pixels to be smoothed will be spaced *smooth_factor* units away from adjacent sets. The situation is illustrated in figures 6 to 8 with a *smooth_factor* of 20 and an *overlap_factor* ranging from one to three. With other values for the *overlap_factor*, there will either be fewer or more intersecting weighting functions in the interval over which the inner product is taken. A different specification of the *overlap_factor* would cause the curves in figure 6 to either widen (by increasing overlap) or narrow (by decreasing overlap).

6.15.2 The spread of the weighting function

With a unit *overlap_factor*, one might be interested in requiring that the height of the Gaussian be reduced to half of its maximum at the intersection points between adjacent curves. Such a constraint will force the variance of the Gaussian to a particular value, hence the

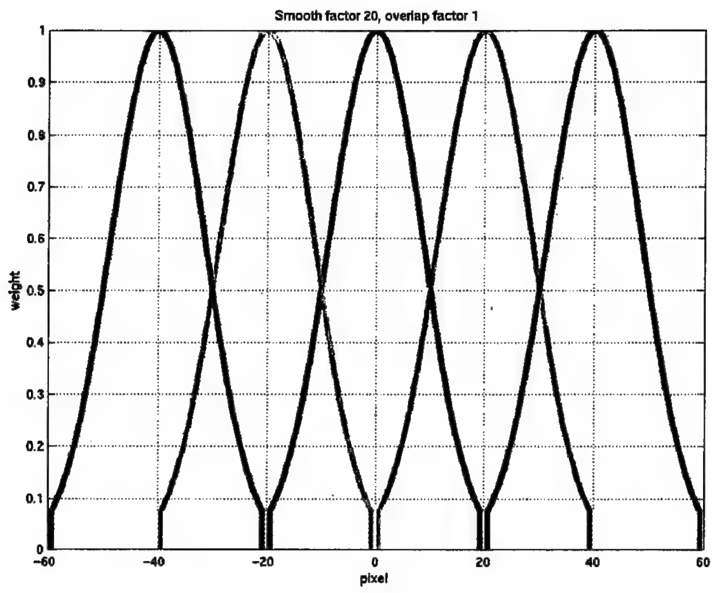


Figure 6: Smoothing and overlap in the atinsar processor: $smooth\_factor = 20$, $overlap\_factor = 1$

appearance of the 2.534 in equation [10]. Indeed, in the language of mathematics, the requirement is derived from

$$\begin{aligned} Ae^{\frac{-(L/2)^2}{2/(L/\gamma)^2}} &= \frac{A}{2}, \\ \frac{-(L/2)^2}{2(\frac{L}{\gamma})^2} &= -\ln 2, \\ \frac{L^2}{4} &= \frac{L^2 \ln 4}{\gamma^2}, \\ \gamma &= \sqrt{4 \cdot \ln 4} \approx 2.35482. \end{aligned} \tag{11}$$

6.15.3 SLC images

The special case of no decimation at all may be directed by specifying an *overlap_factor* of zero and a *smooth_factor* of one. The code will interpret such a set of parameters as a request for a SLC image.

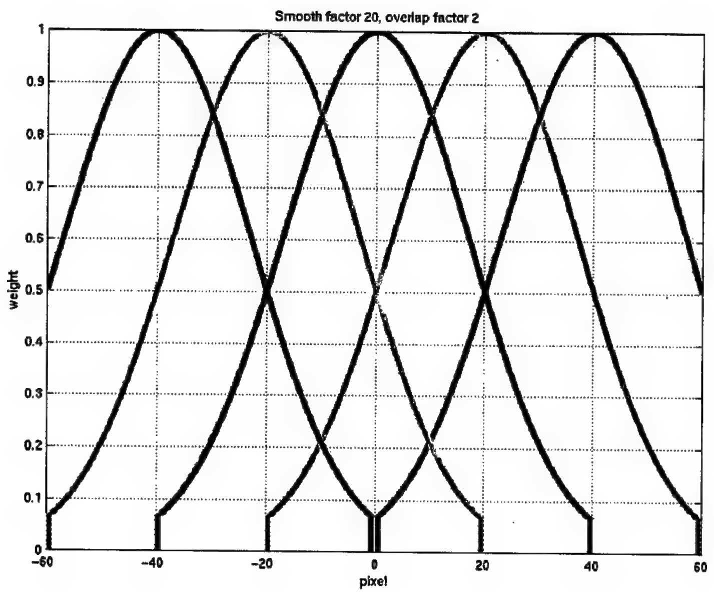


Figure 7: *Smoothing and overlap in the atinsar processor:* $smooth\_factor = 20$, $overlap\_factor = 2$

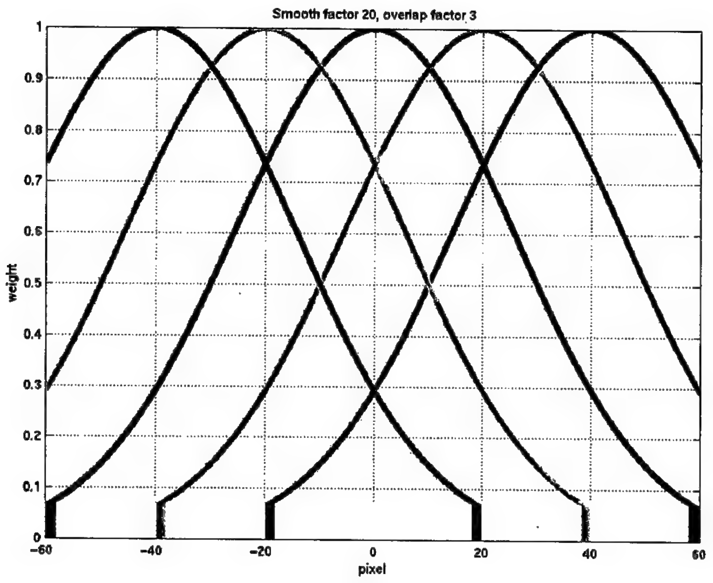


Figure 8: *Smoothing and overlap in the atinsar processor:* $smooth\_factor = 20$, $overlap\_factor = 3$

6.16 Convolution trimming

After convolution, some of the data will not be useful because of the wrap around nature of the FFT. The program clips off the section of data that is no good. The proportion of data that is useful is dependent upon the beam-width at the given range. The beam-width determines the length of the reference function which, in turn, determines the length of data that is spoiled after convolution. A good reference on the end-effects of zero padding convolution is found in [1].

After convolution, the data that is stored in memory will be skewed, as shown in figure 9. The two angled black lines are straight lines that are perpendicular to the aircraft motion vector.

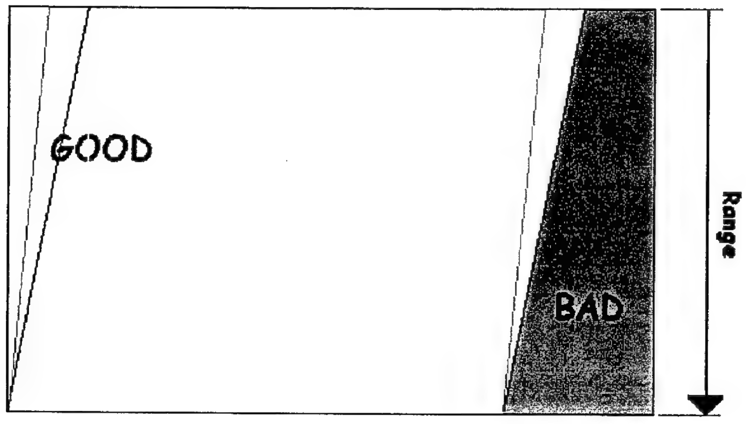


Figure 9: *Convolution Trimming: After convolution, the data appears skewed (the blue parallelogram is actually rectangular in real space).*

7. Notes on the theory

7.1 A full stationary reference function

The original parabolic reference function implemented by the processor was,

$$\theta = \frac{2\pi x^2}{\lambda r_0}, \tag{12}$$

which is just an approximation to the fully hyperbolic reference function given by

$$\theta = \frac{4\pi}{\lambda}\left[\sqrt{x^2 + r_0^2} - r_0\right]. \tag{13}$$

Because of the relatively inexpensive cost of implementing the full reference function, expression [13] was made the reference function.

7.2 Velocity and acceleration characteristics of a moving target

One method by which a moving target may better be investigated is by re-processing a subsection of the raw data in such a way that the matched filter, used for azimuth processing, accounts for target motion. Normally, as in the original processor, the matched filter bandwidth is centered around zero–doppler. One effect in accounting for target motion is a shift in the frequency of the matched filter. There are other more subtle effects as well.

7.3 Approach

The zero–doppler hyperbolic function describing the phase of the matched filter is

$$\theta = \frac{4\pi}{\lambda}\left[\sqrt{x^2 + r_0^2} - r_0\right]. \tag{14}$$

A 4^{th} degree polynomial, $R_4(t)$ is required to describe the phase of the matched filter for a moving target with a constant acceleration, namely,

$$\theta_{mov}(t) = \frac{4\pi}{\lambda}\left[\sqrt{R_4(t)} - r_0\right], \tag{15}$$

where

$$R_4(t) = A_4t^4 + A_3t^3 + A_2t^2 + A_1t + A_0, \tag{16}$$

given that

$$\begin{aligned}
A_4 &= \frac{1}{4}\ddot{x}^2 + \frac{1}{4}\ddot{y}^2 + \frac{1}{4}\ddot{z}^2 = \frac{1}{4}\ddot{x}^2 + \frac{1}{4}\ddot{r}^2, \\
A_3 &= -(v_a - \dot{x})\ddot{x} + \dot{y}\ddot{y} + \dot{z}\ddot{z} = -(v_a - \dot{x})\ddot{x} + \dot{r}\ddot{r}, \\
A_2 &= (v_a - \dot{x})^2 + \dot{y}^2 + y_0\ddot{y} - h\ddot{z} + \dot{z}^2 = (v_a - \dot{x})^2 + \dot{r}^2 + r\ddot{r}, \\
A_1 &= 2y_0\dot{y} - 2h\dot{z} = 2r\dot{r}, \\
A_0 &= y_0^2 + h^2 = r^2.
\end{aligned} \tag{17}$$

In the above equations, v_a denotes the speed of the radar platform and r is the range to the target at time $t = 0$. The situation is as illustrated in figure 10. The frequency shift in the matched filter is defined by the shift in the location of the minimum of expression [15]. Since equation [15] can be written in terms of x and r and their first and second derivatives with respect to time, it is not possible to resolve the vertical motion components from the ground range motion components. Of course, the emitted signal from the aircraft is spherically symmetric, so the inability to distinguish vertical from horizontal motion is not surprising.

7.3.1 Azimuth displacement

Targets with motion may be azimuthally displaced from their true positions. The amount of shift in the position of a target is defined by the time at which equation [15] is minimized. It is sufficient to equate the derivative of $R_4(t)$ to zero and find t

$$0 = 4A_4t^3 + 3A_3t^2 + 2A_2t + A_1. \tag{18}$$

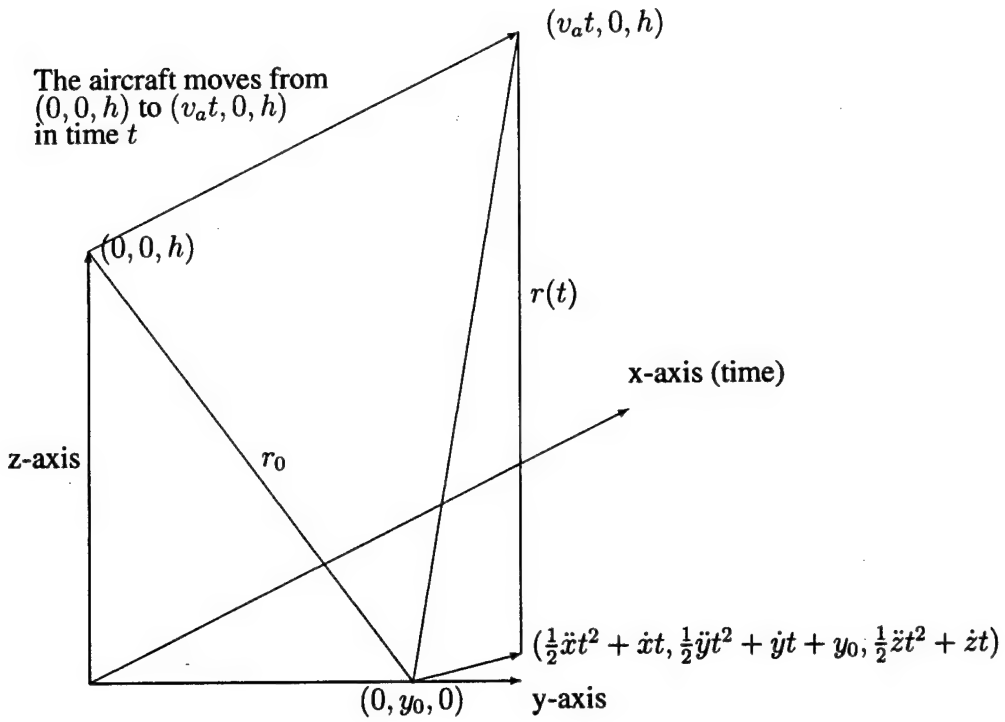
If, for the sake of argument, $\ddot{x} = \ddot{y} = \ddot{z} = 0$, *i.e.* that there are no acceleration terms, then $A_4 = A_3 = 0$, and the above can be simplified to

$$t = \frac{-A_1}{2A_2} \tag{19}$$

$$= \frac{h\dot{z} - y_0\dot{y}}{(v_a - \dot{x})^2 + \dot{y}^2 + \dot{z}^2} \tag{20}$$

The denominator in equation [20], a sum of squares, will be dominated by the aircraft velocity to a first approximation if the target velocity is very much smaller than the aircraft velocity. Thus, equation [20] can be approximated by

$$t = \frac{h\dot{z} - y_0\dot{y}}{v_a^2}, \tag{21}$$



$(v_a t, 0, h)$
The aircraft moves from $(0,0,h)$ to $(v_a t, 0, h)$ in time t
$(0,0,h)$
$r(t)$
x-axis (time)
z-axis
r_0
$(\frac{1}{2}\ddot{x}t^2 + \dot{x}t, \frac{1}{2}\ddot{y}t^2 + \dot{y}t + y_0, \frac{1}{2}\ddot{z}t^2 + \dot{z}t)$
$(0, y_0, 0)$
y-axis

Figure 10: The geometry of a moving target

or, in terms of distance, $x_{disp} = v_a \cdot t$. The displacement will be, approximately,

$$x_{disp} = \frac{h\dot{z} - y_0\dot{y}}{v_a} \tag{22}$$

$$= \frac{-r_0\dot{r}}{v_a} \tag{23}$$

$$= \frac{-r_0\lambda\theta}{4\pi d}, \tag{24}$$

where r_0 and $\dot{r}$ are the range to the target and the radial velocity of the target at zero-doppler, respectively, where we have used $\dot{r} = \lambda\theta v_a/4\pi d$, ($\theta$ is the measured phase of the target, λ is the wavelength, and d is the baseline).

7.3.2 Effects

The most obvious effect of a radial velocity offset is to shift the azimuth section of data that emerges from the processor. In addition to shifting the image, $\dot{r}$ also affects the measured response of a target. The azimuth reference function (for convolution) has not only a phase dependence, but also a cosine envelope structure. Consequently, targets which have higher or lower frequencies than zero–doppler will be discriminated against by the envelope function. The faster the radial velocity of a moving target in the SAR image, the higher its frequency offset from zero–doppler and the weaker its processed response.

7.3.3 Complications of velocity offset processing

There are a couple of ambiguous points to keep in mind when re-processing with a velocity offset. The first is in the interferometric phase as it sweeps past $-\pi$ in either a clockwise or counter-clockwise sense. The second, in the MTI detection software, is defined by the need to be able to distinguish between 2π wraps of the phase such as $-3\pi/4$ and $5\pi/4$. Phase unwrapping is possible because we can get a sense of the direction of motion for the moving targets with the *apriori* knowledge of the offset ground range velocity that was used for processing.

7.3.4 Processing ambiguity

An ambiguity will manifest at around $\dot{y} = 11.98$(m/s), $\dot{z} = 0$(m/s) when the imaging incidence angle is about 50.9 degrees. There are many values for $\dot{y}, \dot{z}$ for which an ambiguity will arise. Indeed, for any value of

$$\dot{r} = \frac{\lambda f_p}{2}, \tag{25}$$

ambiguity will be a concern.

7.3.4.1 Derivation

There are two ways to approach the derivation, either through a frequency approach or a phase approach. We shall use the phase approach.

Imagine the situation where an offset velocity has been applied to the matched filter such that the phase of the matched filter has been shifted and dropped by a certain amount (see the different phase curves in figure 14). Note that the phase of the matched filter always passes through the origin. We now ask the question: Is it possible that the phase at the first sample point is displaced by some multiple of 2π from the case where there is no velocity offset applied? In the case where there is no acceleration and no azimuth velocity offset, consider the difference between the two reference function phases at the point t and equate to $2n\pi$. i.e.,

$$2n\pi = \frac{4\pi}{\lambda}\left(\sqrt{(v_a^2+\dot{r}^2)t^2+2r_0\dot{r}t+r_0^2}-\sqrt{v_a^2t^2+r_0^2}\right),$$
$$\frac{n^2\lambda^2}{4}+n\lambda\sqrt{v_a^2t^2+r_0^2}=\dot{r}^2t^2+2r_0\dot{r}t. \tag{26}$$

With the assumptions that $v_a >> \dot{r}$, $v_at << r_0$, and $n^2\lambda^2/4 \approx 0$, we get

$$\dot{r}^2t^2+2r_0\dot{r}t-n\lambda r_0 \approx 0. \tag{27}$$

Since we are concerned with small times (t), and since r_0 is typically very large, the coefficient of $\dot{r}^2$ is very much smaller than that of $\dot{r}$, so we can make the approximation that equation [27] is mostly linear thus giving

$$\dot{r}=\frac{n\lambda}{2t}. \tag{28}$$

Now, with the choice $n=1$, *i.e.* the first sample point at $t=1/f_p$, the above gives

$$\dot{r}=\frac{f_p\lambda}{2}. \tag{29}$$

The phase difference at all other sample points will also be multiples of 2π. To see that the other points are multiples of 2π as well, we need only recall that around the point $t=0$ the derivative of the phase with respect to time (frequency) is approximately linear. Therefore, if the difference between the zeroth sample point is zero and the difference between the first sample point $t=\frac{1}{f_p}$ is 2π, then the difference between the l^{th} sample point will be $2\pi l$ and the phase will have completely wrapped around at every point in the synthetic aperture. Note that when we say difference in the above, we mean the difference in phase between the zero-offset matched filter and the velocity offset matched filter at the same point in time t.

Figure 11 shows the blind speed in terms of $\dot{y}, \dot{z} = 0$ as a function of range (incidence angle) for line 7, pass 10, Petawawa, 1999.

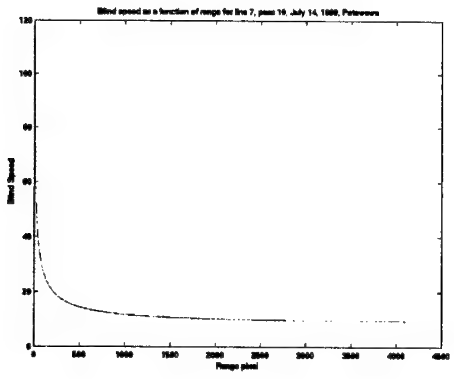


Figure 11: Chart showing the blind speed vs. range. Petawawa, July 14, 1999, line 7, pass 10

7.3.5 Interferometric ambiguity

The expression for the interferometric ambiguity is more easily derived than that for the SAR ambiguity. The radial velocity of a moving target is given by:

$$\dot{r} = \frac{\Delta r}{\Delta t}, \tag{30}$$

where Δt is the time between looks (by the two antennas) and where

$$\Delta r = r_2 - r_1 \tag{31}$$

$$= \frac{\lambda \theta}{4\pi}. \tag{32}$$

An accurate estimate of Δr is only obtained when Δr is such that $-\pi < \theta \leq \pi$. For example, if the real Δr gives $\pi < \theta < 2\pi$, software will measure θ as between $-\pi < \theta < 0$, and the velocity estimate will be wrong. The expression for the wrap velocity is given by:

$$\dot{r} = \frac{\frac{\lambda 2\pi}{4\pi}}{\Delta t} \tag{33}$$

$$= \frac{f_p \lambda}{2}, \tag{34}$$

which is exactly the same expression for the wrap velocity given for the SAR ambiguity. It should be noted that the time between looks is given by $1/f_p$, which is the pulse repetition frequency, and is only valid when the two images have been registered so that the time between looks is, in fact, matched to the prf.

7.4 Frequency domain investigation of the tuned filter

The tuned filter given by equation [15] can be closely approximated by a displaced (from the origin) parabola around the point $x = 0$. In the processor *atinsar1.4*, however, the approximation is not made, and the full reference function is implemented. Only a small section of the matched filter is used for processing. The following figure shows how a plot of the angular frequency of the matched filter (the derivative, with respect to time of equation [15]) is asymptotic to $2v_a f_p \approx 25564$. A spectral histogram of the azimuth lines passing

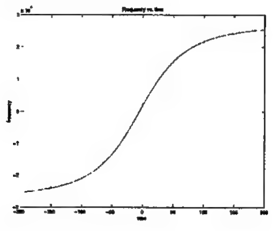


Figure 12: Frequency limits of the matched filter

through a moving target can show how movers are separated in frequency from clutter. Figure 13 shows the frequency spectrum of the target on the Juliet track, line 6, pass 7, July 14, 1999, Petawawa. The task of matching the filter to the target as seen in the frequency spectrum is a function of translation of the matched filter to the frequency band under investigation, and rotation of the filter so as to match the correct azimuth speed. Figures 14 to 15 show how the matched filter depends upon the ground range and the azimuth velocity offsets used.

7.5 Building a reference function that incorporates subpixel azimuth shifts

Part of the creation of the interferogram step requires that one image be sub-pixelly shifted so that the two images appear as though they were taken

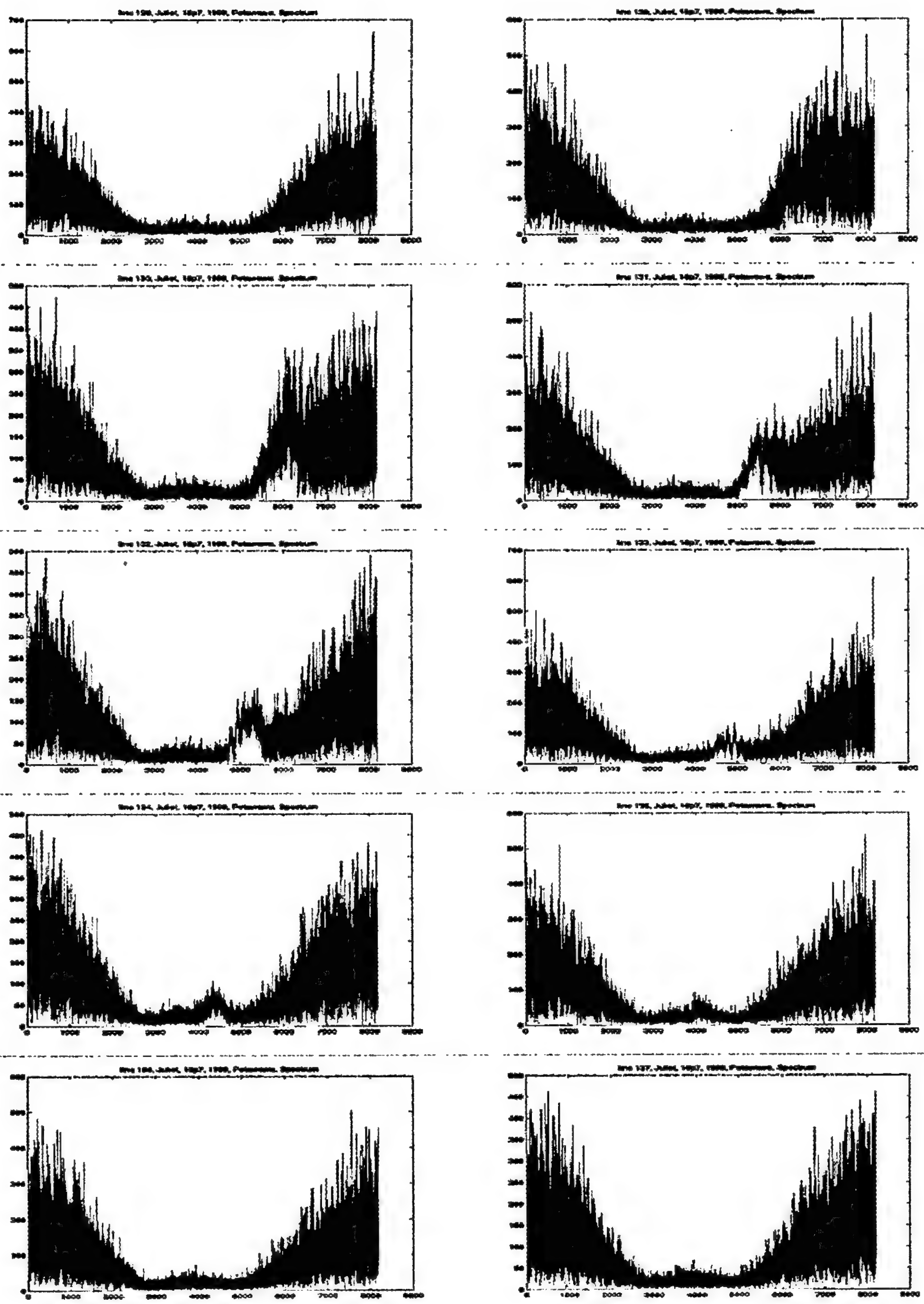


Figure 13: Spectrum of the Juliet target, line 6, pass 7, Petawawa, 1999. The third graph shows the azimuth line that passes through the center of the target.

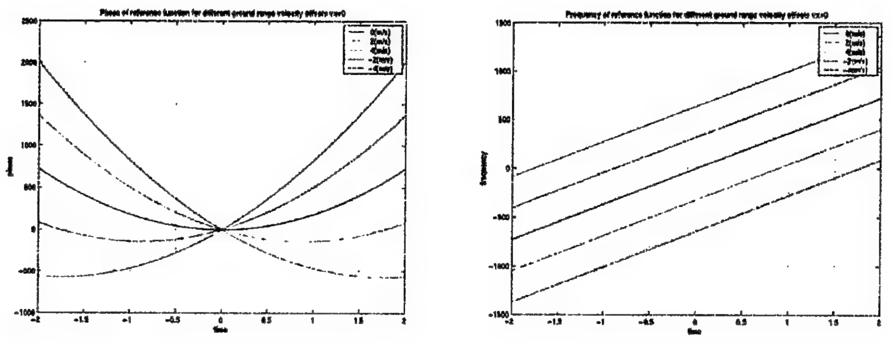


***Figure 14:** Ground range velocity offset variation, $\dot{y}$, in the matched filter*

from the same point in space. In the initial version of the software, a cubic spline method was used to interpolate the sub-pixel values.

A concern was that spline methods might spoil the phase structure of the signal. It was therefore decided to apply Fourier methods to do the sub-pixel shift. In actual fact, the Fourier methods seem to be quicker, and easier to implement. So far, only the azimuth shift is done using Fourier methods, while the range shifts are still done with cubic spline methods. The range shift shall eventually be modified.

Instead of considering an appropriate phase shift to apply to the frequency representation of the signal data, the reference function was simply shifted.

Given a reference function symmetric around the origin

$$r = r(t), \tag{35}$$

the compressed signal is given by:

$$s(t) = \int_{t-L/2}^{t+L/2} f(x)r(x-t)dx, \tag{36}$$

where $f(x)$ is the raw signal data. Thus, a shift in the calculation of $r(t)$, to $r'(t) = r(t + \Delta t)$ will result in a shift in $s(t)$, *i.e.* $s'(t) = s(t + \Delta t)$. Because of the t dependence in the limits of the integral, this will only work if the limits are shifted as well; for subpixel shifts, it shouldn't matter at all. In this way, we can calculate the signal from the fore antenna at time $s(t)$, and the signal from the

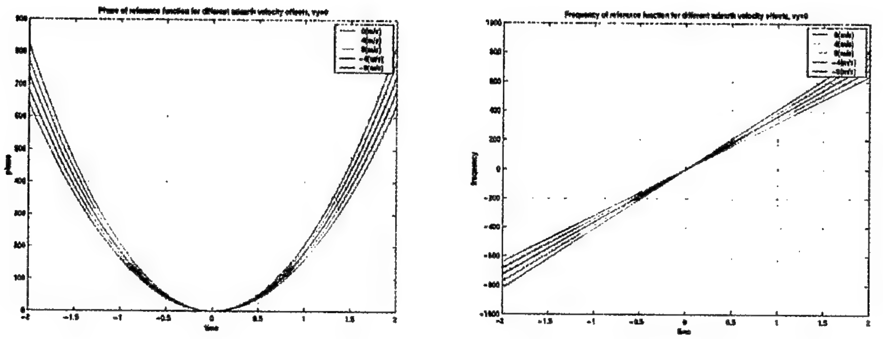


Figure 15: *Azimuth velocity offset variation, $\dot{x}$, in the matched filter*

aft antenna at time $s(t + \Delta t)$ and then calculate the interferogram with the properly registered images.

Another way to think of it is that a spatial shift in the reference function will result in a phase shift in the spectrum of the reference function, which when multiplied by the spectrum of the signal data and brought back into the spatial domain will result in a spatial shift in the detected target.

7.6 Beam splitting

It may be desirable to use only a portion of the beam to process the data. That is, one may wish to only use a section of the reference function defined in the previous section to focus the data, an equivalence with beam splitting.

The reference function in the processor is defined by multiplying a phase function by a weighting function. The phase function defines the history of the signal data over time while the weight function is usually designed to minimize side-lobe contributions. The phase history as recorded by the synthetic aperture defines the azimuth beam. An intersecting observation is that by using only a subsection of the available phase history, the synthetic aperture beam can essentially be split into sections.

Unfortunately, if such a beam splitting scheme is implemented, the resulting image will be geometrically skewed. For example, if only the first section of the beam is used, there will be no point around which the section of beam is symmetric (as it is with the whole beam centered around zero doppler). An equivalence is with a squinted radar beam. Thus, when creating the image, the squint will have to be considered. By calculating the position of the section of the beam as a function of range, appropriate shifts can be applied so that the resulting processed image is not skewed; one method via which such a scheme could be instituted is by that described in section 7.5. Since the azimuth reference function is calculated for each range cell, it will not require significant additional computational time to build an azimuth shift into the reference function.

The compensation for azimuth shift due to squint has not yet been built into the processor.

8. DREO IP

This section lists the developments built into the processor which constitute DREO Intellectual Property (IP). All of the listed IP was inserted into the processor by DREO between January 2000 and November 2000.

1. **Dynamic Reference Function of a Moving Target:** The parabolic approximation to the reference function of a stationary target was upgraded to the flexible, dynamic reference function, without approximation, of a moving target as described in section 7.2.

2. **Azimuth Subpixel Shift Implementation:** It was thought that the phase structure of cubic spline interpolated data might be compromised, so a different (more accurate) method for the azimuth subpixel shifting was developed and implemented. The resulting procedure as described in section 7.5 hastens program execution and improves data integrity.

3. **Beam Splitting:** Although not complete, beam splitting as outlined in section 7.6 will open up additional areas for investigation. The technique of examining the range centroid shift over different sections of beam processing might provide additional information as to the movement characteristics of a target.

4. **MATLAB I/O Library Creation:** The MATLAB interface between the C-programs and MATLAB matrix data was limited to MATLAB version 4 and presented obstacles to the processing scheme not only in that data formats had to be tampered with, but that pre-processing took longer than necessary. The upgrade to MATLAB version 5 format as described in section 5.5 decreased the pre-processing time and eliminated the special steps that had to be taken to ensure that formats were not in conflict. Furthermore, the current upgrade has paved the road to integration with future versions of MATLAB.

5. **MATLAB 5 Upgrade:** As described in section 5.5 the MATLAB routines were upgraded to version 5 so that speedier routines could be used.

6. **IRIX 6.5 Upgrade:** The original software was written for an IRIX 5.x operating system. It has been upgraded to 64-bit, IRIX 6.5 format as is described in section 5.2. Changes were made to the code to enable the 64-bit mode which provides for greater execution speed and greater data handling capability. Once again, an additional bonus is that the current upgrade has facilitated future upgrades.

7. **Execution Scheme:** One change that has made the program much more flexible is the change made to the execution scheme. With the idea of command line arguments as listed in section 5.6, the program has become more dynamic. The flexibility that has been introduced covers flexible reference function definition, output format, execution location, and other options easily found by using the -help command line argument.

8. **Code Scheme:** The code has been greatly cleaned up. Much work has gone into separating the various program tasks into modules. As discussed in section 5.8 the old idea of including source code at the end of the program with a large number of include statements has been modernized to dynamic linking with run time libraries. Some of the advantages of such an upgrade are the clearing of the upgrade path for future improvement, the sharing of the various functions with other programs that need to execute similar tasks, and the reduction in the complexity of the code from a developmental point of view. Included in the new version of the program are a library designed to complete MATLAB I/O operations, a library designed to execute numerical functions like FFT's and detection, and a library in which many utility functions are gathered.

   The following are libraries that were created at DREO

   - *libc_shared.so*
   - *libc_shared64.so*
   - *libc_matio.so*
   - *libc_matio64.so*
   - *libc_numerical.so*
   - *libc_numerical64.so*

9. **Documentation:** By no means the least of tasks undertaken by DREO was the documentation. A massive effort, section 6, was made to expose the inner workings of the various programs and MATLAB functions. A program processing guide has constructed as well as a program compilation guide. Such documentation did not seem to exist beforehand, and should significantly alleviate the difficulty associated with training an individual to not only run the software, but to continue to enhance the software. This document is itself DREO IP.

References

1. William H. Press, Saul A. Teukolsky, William T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, 2nd edition, 1988.

2. Silicon Graphics, *IRIX 6.5 Release Notes*. Silicon Graphics Inc. Internet pulication
http://techpubs.sgi.com/library/tpl/cgi-bin/init.cgi

3. Merrill I. Skolnik, *RADAR HANDBOOK, Second Edition*. McGraw–Hill, Inc., New York, 2nd edition, 1990.

4. Paris Vachon and John Wolfe "The Software-Based Motion Compensation System for the CCRS airborne SARs: Technical Description V2.0" CCRS Internal Document, October, 1991. CCRS, Ottawa.

5. A. Freeman and A. Currie, "Synthetic Aperture RADAR (SAR) Images of Moving Targets". GEC J of Research, Vol. 5, No. 2. pp. 106-115, 1987.

6. J.W.M Campbell, A.L. Gray, K.E. Mattar, J.H. Clarke, and M.W.A. Van der Kooij, "Ocean Surface Feature Detection With the CCRS Along Track InSAR". Can. J. of Remote Sensing, Vol 23, No. 1, March 1997.

7. Giorgio Franceschetti and Riccardo Lanari, *Synthetic Aperture RADAR PROCESSING*. CRC Press, Washington D.C., 1999.

DOCUMENT CONTROL DATA
(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.) DREO	2. SECURITY CLASSIFICATION (overall security classification of the document, including special warning terms if applicable) UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C or U) in parentheses after the title.)

The Convair 580 Along-Track INSAR Processor(U)

4. AUTHORS (Last name, first name, middle initial)

Sikaneta, Ishuwa - Campbell, John - Robson, Mike

5. DATE OF PUBLICATION (month and year of publication of document) August 2001	6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc.) 103	6b. NO. OF REFS (total cited in document) 7

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Memo

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.)
DREO

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant) 5eg11	9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written)
10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.) DREO TM 2001-053	10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor)

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(X) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies; further distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

Unlimited

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

This document summarizes the work done on the along-track interferometric SAR processor loaned to Defence Reseach Establishment Ottawa (DREO) by the Canada Centre for Remote Sensing (CCRS). It outlines the problems encountered with getting the software to work properly, as well as the enhancements that were made to the code. It provides a guide to setting up and using the processor. It details the changes in structure made to the program , and describes some of the theoretical ideas behind changes and enhancements. It provides, finally, a document deliverable to CCRS that fulfulls part of the loan agreement.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Along-track interferometry, GMTI, SAR processing, Convair 580, Motion compensation, Matched filter, SGI libraries, Multi-look processing, detection, estimation.